

APPENDIX

Inventors: Bruce T. Petro, Andrew Cohen and Jason Sulak

Title: ON-LINE SYSTEM FOR CREATING A PRINTABLE PRODUCT

The first part of the paper is devoted to the study of the asymptotic behavior of the solutions of the system (1) as $\epsilon \rightarrow 0$. In the second part, we study the asymptotic behavior of the solutions of the system (1) as $\epsilon \rightarrow 0$.

```
/*=====
=
```

File: scannota.h

\$Header: /Projects/Toolbox/ct/SCANNOTA.H 2 5/30/97 8:4
5a Wmanis \$

Contains: Generalized annotation structure.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```
=====*
```

```
/
#ifndef _H_SCANNOTA
#define _H_SCANNOTA
```

```
#include "sctypes.h"
#include <string.h>
```

```
class scAnnotation {
```

```
public:
```

```
    scAnnotation( UCS2*, int, int, int );
```

```
    scAnnotation();
```

```
    void Set( UCS2 *, int, int, int );
    void Clear( void );
```

```
    UCS2 fCharStr[32];
    Bool fAnnotate;
```

```
// if true
```

```
apply, else clear
```

```
        int                fParaOffset;
        int                fStartOffset;
        int                fEndOffset;
};

/* =====
===== */

inline void scAnnotation::Clear()
{
    fAnnotate                = false;
    fParaOffset              = -1;
    fStartOffset             = -1;
    fEndOffset               = -1;
}

/* =====
===== */

inline scAnnotation::scAnnotation( UCS2 *ch, int paraoffset, int s
tart, int end )
{
    Set( ch, paraoffset, start, end );
}

/* =====
===== */

inline scAnnotation::scAnnotation()
{
    Clear();
}

/* =====
===== */

#endif
```

//<html><pre>

/*****

File: SCAPPINT.H

\$Header: /Projects/Toolbox/ct/SCAPPINT.H 2 5/30/97 8:45a Wmanis \$

Contains: The portable c application interface prototypes

SCENG_XXXXX - messages to the text engine

SCFS_XXXXXX - messages to flowsets

SCCOL_XXXXX - messages to columns

SCSEL_XXXXX - messages to a selection

SCSTR_XXXXX - messages to streams

SCSCR_XXXXX - messages to the scrap

SCHRGX_XXXX - messages to regions

SCCHTS_XXXX - messages to scCharSpecList

SCTSL_XXXXX - messages to scTypeSpecList

SCAPPTXT_xx - messages to scAPPText

SCRDL_XXXXX - messages to scRedisplayList

Written by: Manis

Copyright (c) 1989-1994 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

@doc

/*****

```
#ifndef _H_SCAPPINT
#define _H_SCAPPINT
```

```
#ifdef SCMACINTOSH
#pragma once
#endif
```

```
#include "sctypes.h"
#include "sccharex.h"
```

// [doc] [test]

class stTextImportExport; // in scapptex.h (scTextExchange)

```
class scTypeSpecList; // in scpubobj.h
class scLineInfoList; // in scpubobj.h
class scRedisplList; // in scpubobj.h
class scSpecLocList; // in scpubobj.h
```

class scSet; // see scset.h

class scImmediateRedispl; // in scpubobj.h

class clField; // in sccallbk.h

// TERMINOLOGY


```

/* =====
FLOW SET      a set of linked columns which contain a single stream
COLUMN        an area to flow text into, text MAY extend outside the
               column depending upon constraints, another name for a column
               is a TEXT CONTAINER
STREAM        a set of content units/paragraphs
CONTENT UNIT   a paragraph that contains characters and specs associated
               with the characters
SELECTION      a range of text, each flow set may only contain one selection
*/

```

```

* ===== */

```

```

/*===== SYSTEM LEVEL MESSAGES =====*/
// @func status | SCENG_Init | This must be called before any other
// calls are made into the Toolbox; it initializes and sets
// toolbox behavior. The base error is the number added to
// <t status> errors when exceptions are re-raised across the api.
//
// @parm int | baseError | Value to add to <t status> values
// if we are re-raising exceptions across the API.
//
status scIMPL_EXPORT    SCENG_Init( int baseError = 0 );

```

```

// @func Closes the Composition Toolbox; this releases all memory
// that the Toolbox has allocated. All references into the
// Toolbox become invalid.

```

```

status scIMPL_EXPORT    SCENG_Fini( void );

```

```

// The following three calls are optional. Their use will guarantee that
// the Toolbox can always recover from an out of memory condition,
// given that the application can get it back to a previous state.

```

```

// Holds memory to guarantee that if recomposition fails, we will
// be able to revert to previous state.

```

```

// [ ] [ ]
status scIMPL_EXPORT    SCENG_RetainMemory( void );

```

```

// Informs Toolbox to use retained memory -
// only for use in a recovery operation.

```

```

// [ ] [ ]
status scIMPL_EXPORT    SCENG_UseRetainedMemory( void );

```

```

// Releases retained memory; recomposition has been successfully completed.

```

```

// [ ] [ ]
status scIMPL_EXPORT    SCENG_ReleaseMemory( void );

```

```

/* ===== OBJECT ALLOCATION ===== */
/* ===== */

```

```

// If the application wants to allocate its objects it can do
// so bearing in mind that it must have set up an exception
// handler, otherwise the client may use the following to
// allocate and free the objects. Accessing the data within
// these objects should not present exception problems if the
// correct accessor methods are used.

```

```

/* ===== */
/* ===== SCTYPESPECLIST ===== */
/* ===== */

```

```

// @func Safely allocate.
status scIMPL_EXPORT SCTSL_Alloc(
    scTypeSpecList*& tsl ); // @parm <t scTypeSpecList>

```

```

// @func Safely delete.
status scIMPL_EXPORT SCTSL_Delete(
    scTypeSpecList*& tsl ); // @parm <t scTypeSpecList>

```

```

/* ===== */
/* ===== SCREDISPLIST ===== */
/* ===== */

```

```

// @func Safely allocate.
status scIMPL_EXPORT SCRD_L Alloc(
    scRedisplList*& rdl ); // @parm <c scRedisplList>

```

```

// @func Safely delete.
status scIMPL_EXPORT SCRD_L Delete(
    scRedisplList*& rdl ); // @parm <c scRedisplList>

```

```

/* ===== */
/* ===== SCAPPTXT ===== */
/* ===== */

```

```

// @func Safely allocate.
status scIMPL_EXPORT SCAPPTXT_Alloc(
    stTextImportExport*& atxt ); // @parm <c scAPPText>

```

```

// @func Safely delete.
status scIMPL_EXPORT SCAPPTXT_Delete(
    stTextImportExport* atxt ); // @parm <c scAPPText>

```

```

/* ===== */
/* ===== SCCHARSPECLIST ===== */
/* ===== */

```

```

// @func Safely allocate.
status scIMPL_EXPORT SCCHTS_Alloc(
    scSpecLocList*& cslist, // @parm <c scCharSpecList>
    scStream* stream ); // @parm <c scStream> to associate w
ith.

```

```

// @func Safely delete.
status scIMPL_EXPORT SCCHTS_Delete(
    scSpecLocList*& cslist ); // @parm <c scCharSpecList>

```

```

/* ===== */
/* ===== CONTAINER MESSAGES ===== */
/* ===== */

```

```

// @func Creates a new column/container within the Composition Toolbox universe.
// The appName is simply a notational convenience for the client. It is
// presumed that the client is maintaining some sort of container structure
// and the appName is typically pointing to the clients structure. In all
// conversations with the client we use this name. If it is 0 we will simply
// fill in our name for it.
//
status scIMPL_EXPORT SCCOL_New(

```

```

        APPColumn appName, // @parm <t APPColumn>, the name used by client.
        scColumn*& newID, // @parm <c scColumn> name of object allocated b
y toolbox.
        MicroPoint width, // @parm <t MicroPoint> width of new text contain
er.
        MicroPoint depth ); // @parm <t MicroPoint> depth of new text contain
er.

```

```

// @func Deletes a container, removes itself from the flowset with
// the text in this container simply spilling over into
// others in the flow set.
// If it is the only column associated in the flow set
// then the stream is also deleted.
// To delete the whole flow set start deleting from the last column
// so that the first column (and the stream) will be deleted last.

```

```

status scIMPL_EXPORT SCCOL_Delete(
        scColumn* col, // @parm Name of <c scColumn> to delete.
        scRedisplist* rInfo ); // @parm <c scRedisplist>
// Redisplay info, arg may be zero.

```

```

// @func Links the two columns together using the following logic:
// COL2 must represent the first column in a flow set.
// COL1 may be anywhere within a (distinct) flow set,
// COL2 will be chained in after COL1. The ordering/reordering
// of the streams is as follows: <nl>
// 1. if COL1 has text and COL2 has no text, the text flows from COL1 to COL2
// <nl>
// 2. if COL2 has text and COL1 has no text, the text flows from COL1 to COL2
// <nl>
// 3. if both COL1 and COL2 have text, the text from COL2 is appended
// to the text in COL1 - this may may create some confusion
// on the user's part, so use with care. <nl>
// NOTE!!!!!! ANY SELECTION IN EITHER FLOW SET WILL NO LONGER BE VALID!!!!
// @xref <f SCCOL_Unlink>

```

```

status scIMPL_EXPORT SCCOL_Link(
        scColumn* col1, // @parm Name of first <c scColumn>.
        scColumn* col2, // @parm Name of second <c scColumn>.
        scRedisplist* rInfo ); // @parm Redisplay info, arg may be zero.

```

```

// @func Unlinks a column from its chain. The column passed in becomes
// an empty container, and the stream remains intact. If the column
// is the only column in a chain, it is a no-op.
// <nl>NOTE!!!!!! ANY SELECTION IN THE FLOW SET
// COLUMN WILL NO LONGER BE VALID!!!!
// @xref <f SCCOL_Link>

```

```

status scIMPL_EXPORT SCCOL_Unlink(
        scColumn* col, // @parm Name of <c scColumn> to unlink.
        scRedisplist* rInfo ); // @parm <c scRedisplist>
// Redisplay info, arg may be zero.

```

```

// @func Severs the link between the two columns. The stream is left in
// the first logical container set. The text is left in a
// uncomposed state.<NL>
// NOTE!!!!!! ANY SELECTION IN THE FLOWSET
// COLUMN WILL NO LONGER BE VALID!!!!
//

```

```

status scIMPL_EXPORT SCFS_Split(
        scColumn* col1, // @parm <c scColumn> prior to split.
        scColumn* col2 ); // @parm <c scColumn> after split.

```

```

// @func Resizes a column. If the column has an irregular shape
// (such as a polygon), the width and depth values of the container
// are updated, but no refloing occurs. The width and depth are independent
// of text flow. Width is always the horizontal dimension and depth
// the vertical dimension.
//

```

```
status scIMPL_EXPORT SCCOL_S_Resize(
    scColumn* col, // @parm <c scColumn> to resize.
    MicroPoint width, // @parm New <t MicroPoint> width.
    MicroPoint depth, // @parm New <t MicroPoint> depth.
    scRedisplList* rInfo ); // @parm <c scRedisplList>
// Redisplay info, arg may be zero.

// tests to see if there is more text than is in this column
// this would set the flag to true if:
//     there is text in subsequent linked columns
//     there is unformatted text that will not fit in this column

status scIMPL_EXPORT SCCOL_MoreText(
    scColumn* col,
    Bool& flag );

/*----- STREAM OPERATIONS -----*/

// @func Returns an id to the stream a column/flow set contains.
//
status scIMPL_EXPORT SCCOL_GetStream(
    scColumn* column, // @parm <c scColumn> containing stream.
    scStream*& stream ); // @parm <c scStream> pointer to be filled.

// @func Writes stream to file using the call back write routine.
// @xref <f SCCOL_GetStream>

status scIMPL_EXPORT SCSTR_Write(
    scStream* stream, // @parm <c scStream> to write.
    APPCtxPtr ioctxptr, // @parm <t APPCtxPtr> Abstract file i/o type.
    IOFuncPtr ioFuncPtr ); // @parm <t IOFuncPtr> write function pointer.

// @func Reads stream from file using the call back read routine.
// Use the appropriate calls for file i/o
// <f SCSET_InitRead>, <f SCOBJ_PtrRestore>, <f SCSET_FiniRead>, <f SCCOL_GetStream>

status scIMPL_EXPORT SCSTR_Read(
    scStream*& stream, // @parm Pointer to <c scStream> to be
                        // filled in by Composition Toolbox.
    scSet* enumTable, // @parm Pointer enumeration table.
    APPCtxPtr ioctxptr, // @parm <t APPCtxPtr> Abstract file i/o type.
    IOFuncPtr ioFuncPtr ); // @parm <t IOFuncPtr> write function pointer.

// @func Deletes a stream. If it is associated with a set of linked columns,
// they become a linked set of empty containers. If no columns are
// attached or no redraw is necessary, the scRedisplList will be NULL.
// @xref <f SCCOL_GetStream>
//
status scIMPL_EXPORT SCSTR_Clear(
    scStream* stream, // @parm <c scStream> to delete.
    scRedisplList* rInfo ); // @parm <c scRedisplList>
// Redisplay info, arg may be zero.

// @func Cuts a stream from its associated containers.
// @xref <f SCCOL_GetStream>
//
status scIMPL_EXPORT SCSTR_Cut(
    scStream* stream, // @parm Stream to cut.
    scRedisplList* rInfo ); // @parm <c scRedisplList>
// Redisplay info, arg may be zero.
```

```

// @func Copies a stream, returning a unique id in the second argument
// The copy will not be associated with any containers.
// @xref <f SCCOL_GetStream>
//
status scIMPL_EXPORT      SCSTR_Copy(
                        const scStream* srcStream, // @parm Stream to copy.
                        scStream*&      theCopy ); // @parm The new copy.

// @func Pastes a stream into a container. If the container already has a stream,
// the pasted stream is appended to the existing one. To conserve on
// resources, the streamID is not duplicated. Thus, for multiple
// pastes of one stream, a new copy of the stream must be made for each paste.
//
status scIMPL_EXPORT      SCFS_PasteStream(
                        scColumn*      col,          // @parm <c scColumn> containing stream
                                                    // which theStream will be appended.
                        scStream*      theStream,    // @parm The stream to be appended.
                        scRedisplList* rInfo );      // @parm <c scRedisplList>
                                                    // Redisplay info, arg may be zero.

// Extracts a scContUnit from a scStreamLocation for use with SCSTR_Split
status scIMPL_EXPORT      SCSEL_GetContUnit(
                        scContUnit*& mark,
                        scContUnit*& point,
                        const scSelection* );

/*
This call is used to undo a link. To set this up,
before linking two columns col1 and col2, save references
to their streams, stream1 and stream2, respectively.
When unlinking col1 and col2, call scStreamSplit( stream1, stream2 )
to split the stream into its original two pieces. This call should be
followed with a call to SCPasteStream( col2, stream2, scRedisplList *)
to reset the unlinked column's stream to its original value.
NOTE: both stream1 and stream2 are assumed to be valid (non-NULL).
This call should only be made with reformatting turned off.
*/
/* [ ] [ ]
status scIMPL_EXPORT      SCSTR_Split(
                        scStream*,
                        scContUnit*,
                        scStream*& );

/* @func Compare streams for equality, this tests content and specs
// scSuccess == equality
// @xref <f SCCOL_GetStream>
//
status scIMPL_EXPORT      SCSTR_Compare(
                        const scStream* str1,      // @parm <c scStream>
                        const scStream* str2 );    // @parm <c scStream>

/***** COMPOSITION OPERATIONS & COMPOSITION ERROR RECOVERY *****/
// @func Sets recomposition off or on in the flow set of the indicated column
//
status scIMPL_EXPORT      SCFS_SetRecompose(
                        scColumn*      col,          // @parm Flow set to turn composition off or on
                        Bool            onOff );      // @parm true == on, false = off

// @func Gets the current recomposition state of the flow set.
//
status scIMPL_EXPORT      SCFS_GetRecompose(
                        scColumn*      col,          // @parm Flow set to query.
                        Bool&          onOff );      // @parm Composition flag.

// @func Recomposes the flowset.
//

```

```

status scIMPL_EXPORT    SCFS_Recompose(
                        scColumn*    col,    // @parm Flow set to recompose.
                        scRedisList* rInfo );// @parm <c scRedisList>
                                                // Redisplay info, arg may be zero.

// NOT IMPLEMENTED
// Recompose a portion of the stream in the flowset. Process the flowset
// for the number of ticks indicated. We will process paragraphs
// until time exceeds the ticks
//
// [ ] [ ]
status scIMPL_EXPORT    SCFS_Recompose( scColumn*,
                                        long ticks,
                                        scRedisList* );

// Rebreaks all the lines in a column, with extreme prejudice;
// it ignores the flowset RecomposeHold() setting and processes
// only the indicated column, it will return an error if a prior
// column is uncomposed
//
// [ ] [ ]
status scIMPL_EXPORT    SCCOL_Recompose( scColumn*,
                                        scRedisList* );

// @func Applies the specs in the CharSpecListHandle to the text
// at the locations indicated therein.
// @xref <f SCCOL_GetStream>, <f SCSTR_CHTSList>
status scIMPL_EXPORT    SCSTR_CHTSListSet(
                        scStream*    stream, // @parm <c scStream> to apply scCharSpecList to
                        const scSpecLocList& cslist, // @parm <c scCharSpecList> list of spec
                        scRedisList* rInfo );// @parm <c scRedisList>
                                                // Redisplay info, arg may be zero.
// and locations.

status scIMPL_EXPORT    SCSTR_PARATSListSet( scStream*    stream,
                        const scSpecLocList& cslist,
                        scRedisList* rInfo );

// ===== CONTAINER CONSTRAINT OPERATIONS =====/

// These routines set containers to be flexible in the horizontal or
// vertical dimensions. A flexible container varies in size with its contents.
// A vertically flexible container varies with the number of lines;
// it grows until it reaches either the last character in the stream or
// a column break. A horizontally flexible container varies with the width
// of its widest line; it grows until the end of the paragraph or a hard
// return.
//
// NOTE: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// Container FLEXIBILITY and IRREGULAR shapes are MUTUALLY EXCLUSIVE!!!!!!
// Setting a container to be flexible automatically clears any
// irregular shape associated with the column.
//

// @func Turns on vertical flexibility.
// @xref <f SCCOL_GetVertFlex>, <f SCCOL_ClearVertFlex>

status scIMPL_EXPORT    SCCOL_SetVertFlex(
                        scColumn*    col,    // @parm <c scColumn> to set flex.
                        scRedisList* rInfo );// @parm <c scRedisList>
                                                // Redisplay info, arg may be zero.

// @func Turns off vertical flexibility.
// @xref <f SCCOL_SetVertFlex>, <f SCCOL_GetVertFlex>

status scIMPL_EXPORT    SCCOL_ClearVertFlex(
                        scColumn*    col,    // @parm <c scColumn> to clear flex.

```

```

scRedisplList* rInfo );// @parm <c scRedisplList>
                        // Redisplay info, arg may be zero.

```

```

// @func Gets the vert flex attribute of the column.
// @xref <f SCCOL_SetVertFlex>, <f SCCOL_ClearVertFlex>

```

```

status scIMPL_EXPORT SCCOL_GetVertFlex(
    scColumn* col, // @parm <c scColumn> to get attribute from.
    Bool& onOff ); // @parm Vertical flex attribute, true
                  // equals on, false off.

```

```

// @func Turns on horizontal flexibility.
// @xref <f SCCOL_GetHorzFlex>, <f SCCOL_ClearHorzFlex>

```

```

status scIMPL_EXPORT SCCOL_SetHorzFlex(
    scColumn* col, // @parm <c scColumn> to set flex.
    scRedisplList* rInfo );// @parm <c scRedisplList>
                        // Redisplay info, arg may be zero.

```

```

// @func Turns off horizontal flexibility.
// @xref <f SCCOL_SetHorzFlex>, <f SCCOL_GetHorzFlex>

```

```

status scIMPL_EXPORT SCCOL_ClearHorzFlex(
    scColumn* col, // @parm <c scColumn> to clear flex.
    scRedisplList* rInfo );// @parm <c scRedisplList>
                        // Redisplay info, arg may be zero.

```

```

// @func Gets the horizontal flex attribute of the column.
// @xref <f SCCOL_SetHorzFlex>, <f SCCOL_ClearHorzFlex>

```

```

status scIMPL_EXPORT SCCOL_GetHorzFlex(
    scColumn* col, // @parm <c scColumn> to get attribute from.
    Bool& onOff ); // @parm Horizontal flex attribute, true
                  // equals on, false off.

```

```

// @func Gets the direction of lines in the container,
// and of characters in the lines.
// @xref <f SCCOL_SetFlowDirection>

```

```

status scIMPL_EXPORT SCCOL_GetFlowDirection(
    scColumn* col, // @parm <c scColumn> to query.
    scFlowDir& fd ); // @parm <c scFlowDir> of column.

```

```

// @func Sets the direction of lines in the container,
// and of characters in the lines.
// @xref <f SCCOL_GetFlowDirection>

```

```

status scIMPL_EXPORT SCCOL_SetFlowDirection(
    scColumn* col, // @parm <c scColumn> to set.
    const scFlowDir& fd ); // @parm <c scFlowDir> value to
                          // set of column.

```

```

/***** POLYGON CONTAINER SHAPE OPERATIONS *****/

```

```

#if defined( scColumnShape )

```

```

// The application may create polygons in any way that it sees fit.
// They are then passed into the Toolbox and text flows into them
// using an even-odd area fill algorithm.
//

```

```

// Pastes in a set of vertices to be added to the column's current

```

```

// vertex list.
//
// [ ] [ ]
status scIMPL_EXPORT      SCCOL_PastePoly( scColumn*,
                                           const scVertex*,
                                           scRedisplist* );

// Extracts a copy of the polygon applied to this column,
// causing no recomposition. Useful for editing the polygon.
//
// [ ] [ ]
status scIMPL_EXPORT      SCCOL_CopyPoly( scColumn*,
                                           scVertex*& );

// Clears the polygon applied to this column.
//
// [ ] [ ]
status scIMPL_EXPORT      SCCOL_ClearPoly( scColumn*,
                                           scRedisplist* );

/*----- REGION RUN-AROUND OPERATIONS -----*/

// Regions are high precision descriptions of arbitrary shapes.
// They are useful for representing irregularly shaped containers,
// or for enabling text to run around objects intersecting a given container.
// The application may use them to represent containers
// that have been modified by intersection with other page objects.
//
// @func Applies a region as the shape of this column.
// The region is assumed to be in local coordinates.
//
status scIMPL_EXPORT      SCCOL_PasteRgn(
    scColumn*      col,      // @parm <c scColumn> to apply region to.
    const HRgnHandle rgn,    // @parm Region to apply.
    scRedisplist*  rInfo );  // @parm <c scRedisplist>
                             // Redisplay info, arg may be zero.

//
// @func Extracts a copy of the region applied to this column,
// causing no recomposition.
//
status scIMPL_EXPORT      SCCOL_CopyRgn(
    scColumn*      col,      // @parm <c scColumn> with region
    HRgnHandle& rgn );      // @parm <t HRgnHandle> the region copy.

//
// @func Clears the region belonging to this column.
//
status scIMPL_EXPORT      SCCOL_ClearRgn(
    scColumn*      col,      // @parm <c scColumn> to clear region.
    scRedisplist*  rInfo );  // @parm <c scRedisplist>
                             // Redisplay info, arg may be zero.

// The following functions operate on regions.

// @func Create a new region.
//
status scIMPL_EXPORT      SCHRGN_New(
    HRgnHandle& newRgn,      // @parm <t HRgnHandle>
    MicroPoint sliverSize); // @parm Sliver size.

// @func What sliver size is a region using.
status scIMPL_EXPORT      SCHRGN_SliverSize(

```



```

        const HRgnHandle rgn,          // @parm HRgnHandle>
        MicroPoint&      sliverSize); // @parm Size of sliver.

// @func Dispose a region.

status scIMPL_EXPORT      SCHRGD_Dispose(
        HRgnHandle disRgn );          // @parm <t HRgnHandle> to dispose.

// @func Make a region empty, remove all slivers.

status scIMPL_EXPORT      SCHRGD_SetEmpty(
        HRgnHandle emptyRgn );        // @parm <t HRgnHandle> to empty.

// @func Is a region empty.
// @rdesc scSuccess == empty region

status scIMPL_EXPORT      SCHRGD_Empty(
        const HRgnHandle emptyRgn );  // @parm <t HRgnHandle> to test.

// @func Compare to regions for equality.
// @rdesc scSuccess == equality

status scIMPL_EXPORT      SCHRGD_Equal(
        const HRgnHandle rgn1,        // @parm <t HRgnHandle>.
        const HRgnHandle rgn2 );      // @parm <t HRgnHandle>.

// @func Determine if point is in region.
// @rdesc scSuccess == equality
//
status scIMPL_EXPORT      SCHRGD_PtIn(
        const HRgnHandle   rgn,        // @parm <t HRgnHandle> to test.
        const scMuPoint&   pt );       // @parm <c scMuPoint>

// @func Make a region rectangular.
//
status scIMPL_EXPORT      SCHRGD_Rect(
        HRgnHandle   rng,      // @parm Region to apply rect.
        const scXRect& rect ); // @parm <c scXRect>

// @func Make a region from a set of vertices ( closed polygons(s) ).
// The polygon must have both a horizontal and vertical dimension
// (e.g. it must have interior space)
//
status scIMPL_EXPORT      SCHRGD_Poly(
        HRgnHandle   rng,      // @parm Region to apply polygon.
        const scVertex* polys ); // @parm <c scVertex> Polygon(s) description.

// @func Copy a region.
//
status scIMPL_EXPORT      SCHRGD_Copy(
        HRgnHandle   dstRgn,    // @parm The copy.
        const HRgnHandle srcRgn ); // @parm To copy.

// @func Translate a region.

status scIMPL_EXPORT      SCHRGD_Translate(
        HRgnHandle   rgn,      // @parm Region to inset.
        MicroPoint   x,        // @parm Horizontal translation.
        MicroPoint   y );      // @parm Vertical translation.

// @func Inset a region.

status scIMPL_EXPORT      SCHRGD_Inset(
        HRgnHandle   rgn,      // @parm Region to inset.
        MicroPoint   h,        // @parm Horizontal size change.
        MicroPoint   v );      // @parm Vertical size change.

// @func Is this rect contained within the region.

status scIMPL_EXPORT      SCHRGD_RectIn(
        const HRgnHandle rgn,    // @parm Region to test.

```

```
const scXRect& rect ); // @parm <c scXRect>
```

```
// Perform Boolean operations on regions, the 3rd arg may be one of the
// 2 original regions, in that case it will replace the contents of after the
// operation is complete.
```

```
// @func Performs an intersection of two regions, placing the intersection
// in a third region.
```

```
status scIMPL_EXPORT SCHRGN_Sect(
    const HRgnHandle r1, // @parm <t HRgnHandle>
    const HRgnHandle r2, // @parm <t HRgnHandle>
    HRgnHandle intersection ); // @parm <t HRgnHandle>,
                                // the intersection of r1 & r2.
```

```
// @func Performs the union of two regions, placing the union in a third region.
```

```
status scIMPL_EXPORT SCHRGN_Union(
    const HRgnHandle r1, // @parm <t HRgnHandle>
    const HRgnHandle r2, // @parm <t HRgnHandle>
    HRgnHandle rUnion ); // @parm <t HRgnHandle>,
                          // the union of r1 & r2.
```

```
// @func Performs a difference of two regions, placing the diff
// in a third region.
```

```
status scIMPL_EXPORT SCHRGN_Diff(
    const HRgnHandle r1, // @parm <t HRgnHandle>
    const HRgnHandle r2, // @parm <t HRgnHandle>
    HRgnHandle difference ); // @parm <t HRgnHandle>,
                              // the difference of r1 & r2.
```

```
// @func Performs an xor of two regions, placing the result in a third region.
```

```
status scIMPL_EXPORT SCHRGN_Xor(
    const HRgnHandle r1, // @parm <t HRgnHandle>
    const HRgnHandle r2, // @parm <t HRgnHandle>
    HRgnHandle xor ); // @parm <t HRgnHandle>,
                      // the xor of r1 & r2.
```

```
#endif
```

```
/*===== RENDERING =====*/
```

```
// @func Renders/draws that part of the column lying within the
// given rect. The scXRect is in local coordinates. The Toolbox
// then calls back to the client using <f APPDrawStartLine>,
// <f APPDrawString>, & <f APPDrawEndLine>, passing the <t APPDrwCtx>
// through. This call and <f SCCOL_UpdateLine> are the only two calls
// that cause glyphs to be drawn. ALL DRAWING OF TOOLBOX CONTAINERS
// HAPPENS AT THE BEHEST OF THE CLIENT.
// @xref <k APPDrawStartLine>, <k APPDrawString>, & <k APPDrawEndLine>
```

```
status scIMPL_EXPORT SCCOL_Update(
    scColumn* col, // @parm <c scColumn> to draw
    const scXRect& clipRect, // @parm Clip rect.
    APPDrwCtx dc ); // @parm Drawing context.
```

```
/*===== CONTAINER & LINE EXTENTS =====*/
```

```
// @func Queries ink extents of the column. The extents returned are
// the maximum bounding box of the maximum character extents expressed
```

```

// in the column's coordinate system.
//
status scIMPL_EXPORT SCCOL_QueryInkExtents(
    scColumn* col, // @parm <c scColumn> to query.
    scXRect& inkExtents ); // @parm <t scXRect>

// @func Queries margins of the column. Returns the actual size of the
// container, rather than the extents. Only meaningful for rectangular
// containers. Bear in mind that the ink of text may extend outside of the
// container.
//
status scIMPL_EXPORT SCCOL_QueryMargins(
    scColumn* col, // @parm <c scColumn> to query.
    scXRect& margins ); // @parm <t scXRect>

// Queries positions of the column's lines. For each line, it queries
// the baseline, the extents, and the character characteristics at
// the start (x height, ascender height, descender height, cap height, etc. ).
// The last two parameters indicate the number of lines and whether text
// overflows the column.
// Typically used for alignment purposes. Positions are in the container's
// local coordinates.
//
// [ ] [ ]
status scIMPL_EXPORT SCCOL_LinePositions( scColumn*,
    scLineInfoList*,
    long&,
    Bool& );

// @func Queries the depth of the column. Useful for determining the depth
// of irregularly shaped columns.
status scIMPL_EXPORT SCCOL_Size(
    scColumn* col, // @parm <c scColumn>
    scSize& size ); // @parm <c scSize>

===== SCRAP CONVERSION =====

// These routines provide a means of converting between the Toolbox
// world and the outside world, typically a conversion into the
// lowest common denominator -- text.

// @func Converts Toolbox scrap in the scScrapPtr to global scrap and
// stores it in the given Handle, which should be passed in as a
// valid handle. The Handle then contains
// a "C" string. This is one case where the Toolbox will not free
// the new structure it creates.
//
status scIMPL_EXPORT SCSCR_ConToSys(
    scScrapPtr scrap, // @parm <c scScrapPtr>
    SystemMemoryObject& memobj ); // @parm <c SystemMemoryObject>

// @func Converts global scrap in the Handle to Toolbox scrap
// and places it in the given scScrapPtr, putting it on
// the internal Toolbox clipboard. The handle should contain
// a "C" string, so that when 0 is encountered, we stop reading.
//
status scIMPL_EXPORT SCSCR_SysToCon(
    scScrapPtr& scrap, // @parm <c scScrapPtr>
    const scChar* stringscrap, // @parm Null terminated "C" string
    // from system scrap.

```

```

TypeSpec ts ); // @parm <t TypeSpec> to apply to string.

```

```

// @func Frees the scScrapPtr.

```

```

//
status scIMPL_EXPORT SCSCR_Free(
    scScrapPtr scrap ); // @parm <c scScrapPtr>

```

```

// @func Returns the list of specs referenced by the contents of the scScrapPtr.

```

```

//
status scIMPL_EXPORT SCSCR_TsList(
    scScrapPtr scrap, // @parm <c scScrapPtr>
    scTypeSpecList& tslist ); // @parm <c scTypeSpecList>

```

```

// @func Writes the contents of the scScrapPtr to disk.

```

```

status scIMPL_EXPORT SCSCR_Write(
    scScrapPtr scrapPtr, // @parm <c scScrapPtr>
    APPCtxPtr ctxPtr, // @parm <t APPCtxPtr>
    IOFuncPtr iofuncp ); // @parm <t IOFuncPtr>

```

```

// @func Reads from disk into the scScrapPtr.

```

```

// Scrap to be read must be a column.

```

```

// @xref <f SCSET_InitRead>

```

```

status scIMPL_EXPORT SCSCR_ReadCol(
    scScrapPtr& scrap, // @parm <c scScrapPtr>
    scSet* enumtable, // @parm <c scSet>
    APPCtxPtr ctxPtr, // @parm <t APPCtxPtr>
    IOFuncPtr readFunc ); // @parm <t IOFuncPtr>

```

```

// @func Reads from disk into the scScrapPtr, using the call back

```

```

// read routine, which should conform to the same header as above.

```

```

// Scrap to be read must be a stream.

```

```

status scIMPL_EXPORT SCSCR_ReadStream(
    scScrapPtr& scrap, // @parm <c scScrapPtr>
    scSet* enumtable, // @parm <c scSet>
    APPCtxPtr ctxPtr, // @parm <t APPCtxPtr>
    IOFuncPtr readFunc ); // @parm <t IOFuncPtr>

```

```

/***** TYPE SPECIFICATION *****/

```

```

// @func Informs the Toolbox that the TypeSpec has been changed and

```

```

// tells it what action needs to be taken to respond accordingly.

```

```

// The Toolbox will recompose, rebreak, or repaint (as instructed

```

```

// by SpecTask) and report back on damage. SpecTasks may be ORed

```

```

// together to indicate multiple tasks. The host application

```

```

// can derive the tasks by calling the function SpecTaskCalculate,

```

```

// located in the source module (delivered with the Toolbox)

```

```

// sc_spchg.cpp.

```

```

status scIMPL_EXPORT SCENG_ChangedTS(
    TypeSpec ts, // @parm <t TypeSpec> has changed and text n
    eeds to be

```

```

eSpecTask      specTask, // reformat to reflect the change.
// @parm <t eSpecTask> tells the toolbox
// what action to take to repair the change.
scRedisplList* rInfo ); // @parm <c scRedisplList>
// Redisplay info, arg may be zero.

```

```

// @func Gets a list of TypeSpecs in a column.
// @xref <f SCSTR_TSLIST>, <f SCSEL_TSLIST>

```

```

status scIMPL_EXPORT SCCOL_TSLIST(
    scColumn*      col, // @parm <c scColumn> to query.
    scTypeSpecList& tslist ); // @parm <c scTypeSpecList> contains
// a list of specs used.

```

```

// @func Gets a list of TypeSpecs in a stream. When working with linked columns,
// this is more efficient than iteratively calling SCCOL_TSLIST.
// @xref <f SCCOL_TSLIST>, <f SCSEL_TSLIST>

```

```

status scIMPL_EXPORT SCSTR_TSLIST(
    scStream*      col, // @parm <c scStream> to query.
    scTypeSpecList& tslist ); // @parm <c scTypeSpecList> contains
// a list of specs used.

```

```

// @func Gets a list of ParaTypeSpecs in a stream.
// @xref <f SCCOL_TSLIST>, <f SCSEL_TSLIST>

```

```

status scIMPL_EXPORT SCSTR_ParATSLIST(
    scStream*      col, // @parm <c scStream> to query.
    scTypeSpecList& tslist ); // @parm <c scTypeSpecList> contains
// a list of specs used.

```

```

// @func Gets a list of TypeSpecs in a selection.
// @xref <f SCCOL_TSLIST>, <f SCSTR_TSLIST>

```

```

status scIMPL_EXPORT SCSEL_TSLIST(
    scSelection*   sel, // @parm <c scSelection>.
    scTypeSpecList& tslist ); // @parm <c scTypeSpecList> contains
// a list of specs used.

```

```

status scIMPL_EXPORT SCSEL_ParATSLIST(
    scSelection*   sel, // @parm <c scSelection>.
    scTypeSpecList& tslist ); // @parm <c scTypeSpecList> contains
// a list of specs used.

```

```

// @func Gets a list of the (TypeSpec, character location) pairs
// representing the TypeSpec runs of the stream.

```

```

status scIMPL_EXPORT SCSTR_CHTSLIST(
    scStream*      stream, // @parm <c scStream> to query.
    scSpecLocList& cslist ); // @parm <c scCharSpecList> has
// positions of all specs contained.

```

```

status scIMPL_EXPORT SCSTR_ParATSLIST( scStream*      stream,
                                         scSpecLocList& cslist );

```

```

// @func Gets a list of the (TypeSpec, character location) pairs
// representing the TypeSpec runs of the selection.
// CAUTION: the ends of paragraphs are marked by NULL specs,
// so depending on how many paragraphs the selection traverses,
// there may be multiple NULL specs contained in the list.
// These NULL specs are not terminators of the list;
// rely upon the CharSpecListHandle's count field for the
// accurate number of structures in the list.
//

```

```

status scIMPL_EXPORT SCSEL_CHTSLIST(
    scSelection*   sel, // @parm <c scSelection> to query.

```

```

        scSpecLocList& cslist ); // @parm <c scSpecList> has
                                   // positions of all specs contained.

status scIMPL_EXPORT    SCSEL_PARATSList( scSelection* sel,
                                           scSpecLocList& cslist );

// @func Counts the characters in a stream. This
// does not represent an exact count for file i/o
// of characters written out.
//

status scIMPL_EXPORT    SCSTR_ChCount(
        scStream* str, // @parm <c scStream> to query.
        long& chCount ); // @parm Characters in stream.

#ifdef scFlowJustify

// @func Sets the vertical justification attributes for the column to be
// flush top (no justification), flush bottom, centered, justified,
// or force justified.
//
status scIMPL_EXPORT    SCCOL_FlowJustify(
        scColumn* col, // @parm <c scColumn>
        eVertJust vj ); // @parm <t eVertJust>

// @func Sets the depth of a vertically flexible column and
// vertically justifies it. This is the only way to vertically
// justify a vert flex column. It should not be used with columns
// that are not vert flex.
// @xref <f SCCOL_SetVertFlex>

status scIMPL_EXPORT    SCCOL_SetDepthNVJ(
        scColumn* col, // @parm <c scColumn>
        MicroPoint depth, // @parm Depth to VJ to.
        scRedisplList* rInfo ); // @parm <c scRedisplList>
                                   // Redisplay info, arg may be zero.

#endif /* scFlowJustify */

/*===== CONTENT CUT; COPY, PASTE & CLEAR =====*/

// These routines move text in and out of the Toolbox,
// with filters converting text as necessary.
//

// @func Appends the text contained in the APPText to the end
// of the stream associated with the scColumn* (formerly SReadAPPText).
// @xref <f SCSTR_GetAPPText>

status scIMPL_EXPORT    SCFS_PasteAPPText(
        scColumn* col, // @parm <c scColumn> in flow set to paste
                        // the text.
        stTextImportExport& appText, // @parm <c scAPPText> contains "mar
ked up" text.
        scRedisplList* rInfo ); // @parm <c scRedisplList>
                                   // Redisplay info, arg may be zero.

status scIMPL_EXPORT    SCSEL_PasteAPPText(

```

```

        scSelection*      sel,          // @parm <c scColumn> in flow set to pas
te                                     // the text.
        stTextImportExport&  appText, // @parm <c scAPPText> contains "marked
up" text.
        scRedisplList*      rInfo );   // @parm <c scRedisplList>
                                       // Redisplay info, arg may be zero.

```

```

// @func Returns a copy of the given stream in APPText&.
// @xref <f SCFS_PasteAPPText>

```

```

status scIMPL_EXPORT  SCSTR_GetAPPText(
        scStream*      str,          // @parm <c scStream> to get "marked up" tex
t from.
        stTextImportExport&  appText ); // @parm <c scAPPText> contains "mar
ked up" text.

status scIMPL_EXPORT  SCSEL_GetAPPText(
        scSelection*      str,          // @parm <c scStream> to get "marked
up" text from.
        stTextImportExport&  appText ); // @parm <c scAPPText> contains "mar
ked up" text.

```

```

/*===== CONTENT I/O =====*/

```

```

// These routines read and write ASCII text files with mark-up. */

```

```

// @func Imports Latin-1 text -- adds the contents of the text file
// to the column using the TypeSpec as the default text specification.
// The call back IO function should conform to the header:

```

```

status scIMPL_EXPORT  SCFS_ReadTextFile(
        scColumn*      col,          // @parm <c scColumn>
        TypeSpec        spec,        // @parm <t TypeSpec>
        APPCtxPtr       ctxp,        // @parm <t APPCtxPtr>
        IOFuncPtr       read,        // @parm <t IOFuncPtr>
        scRedisplList*  rInfo );     // @parm <c scRedisplList>
                                       // Redisplay info, arg may be zero.

```

```

// @func Exports text -- writes the stream to the text file.

```

```

status scIMPL_EXPORT  SCSTR_WriteTextFile(
        scStream*      stream, // @parm <c scStream>
        APPCtxPtr       ctxp,  // @parm <t APPCtxPtr>
        IOFuncPtr       write ); // @parm <t IOFuncPtr>

```

```

// @func Reads an Latin-1 text file and returns a scrap handle to it.
// This is useful for importing text and pasting it into a stream
// at an insertion point. The call back IO function should
// conform to the same header as above.
//

```

```

status scIMPL_EXPORT  SCSCR_TextFile(
        scScrapPtr&     scrapP, // @parm <t scScrapPtr>
        APPCtxPtr       ctxp,    // @parm <t APPCtxPtr>
        IOFuncPtr       read );  // @parm <t IOFuncPtr>

```

```

/*===== FILE I/O =====*/

```

```
// These routines read and write the Toolbox structures to disk. */
```

```
// @func Tells the Toolbox that the application is about to
// commence writing structures out. This zeros the enumeration
// count of all objects in the Toolbox within this context
//
status scIMPL_EXPORT    SCTB_ZeroEnumeration( void );
```

```
// @func Tells the object to enumerate itself.
```

```
//
status scIMPL_EXPORT    SCOBJ_Enumerate(
                        scTBObj*    obj,
                        long&        ecount );
```

```
// @func Stores the structures for this object. Streams and linked columns
// are by default written out with the first column; writes to columns
// that are not the first column of a stream are no-ops.
```

```
//
status scIMPL_EXPORT    SCCOL_Write(
                        scColumn*    col,      // @parm <c scColumn>
                        APPCtxPtr    ctxp,     // @parm <t APPCtxPtr>
                        IOFuncPtr    write );// @parm <t IOFuncPtr>
```

```
// @func Tells the Toolbox that the application is about to
// commence reading structures in. This allocates an enumeration
// structure of class scSet, when the file was written out the
// client probably should have noted the enumeration count, by doing this
// we may allocate enough members of the enumeration structure
// to at least guarantee that there will be no failure in inserting
// members into the enumeration structure
```

```
status scIMPL_EXPORT    SCSET_InitRead(
                        scSet*& enumerationTable,
                        // @parm Pointer to enum table
                        // that Composition Toolbox will
                        // allocate - pre allocating the
                        // number of slots indicated to
                        // minimize memory failures on
                        // file i/o.
                        long    preAllocationCount ); // @parm Preallocate this many slots
                                                    // in the enum table.
```

```
// @func Lets the Toolbox know the application is finished reading
// so it can free structures for restoring pointers and will
// recompose everything that needs recomposition.
```

```
//
status scIMPL_EXPORT    SCSET_FiniRead(
                        scSet*        enumTable, // @parm This table will be freed.
                        scRedisplList* rInfo ); // @parm <c scRedisplList>
                                                    // Redisplay info, arg may be zero.
```

```
// @func Read this column.
```

```
//
status scIMPL_EXPORT    SCCOL_Read(
                        APPColumn    appcol,    // @parm Client's <t APPColumn> to
                        // associate with this column.
                        scColumn*&    col,      // @parm <c scColumn>
                        scSet*        enumTable, // @parm <c scSet>
                        APPCtxPtr    ioctxptr,   // @parm <t APPCtxPtr> Abstract file i/o type.
                        IOFuncPtr    ioFuncPtr );// @parm <t IOFuncPtr> write function pointer.
```

```
// @func Tells the object to restore its pointers. This
// function relies upon <f APPDiskIDToPointer>.
```

```
//
```



```

status scIMPL_EXPORT      SCOBJ_PtrRestore(
                           scObj*      obj,          // @parm Restore this objects pointers.
                           scSet*      enumTable );   // @parm Use this enumtable.

```

```

// @func Prior to calling SCOBJ_PtrRestore the client may want to abort
// the action for some reason. In that case call the following
// and all objects that have been read in, but have not had
// their pointers restored will be deleted, including the
// the enumeration table (scSet)
//

```

```

status scIMPL_EXPORT      SCSET_Abort(
                           scSet*& enumTable );      // @parm <c scSet>

```

```

// @func Gives the size of a Toolbox object that will be written to disk.
// The first column in a flow set will contain the content/sctream.
//

```

```

status scIMPL_EXPORT      SCEExternalSize(
                           scColumn*   col,          // @parm <c scColumn>
                           long&        bytes );      // @parm Disk bytes.

```

```

/*===== SELECTION MESSAGES =====*/

```

```

// @func Forces the initial selection within an empty container by creating
// an initial stream. Two conditions present interesting error conditions
// with this call.
// <nl>1. If the formatted text cannot fit into the container a
// scERRstructure is returned.
// <nl>2. If the container is not the first in a flow set then
// scERRlogical is returned.
// <nl>The client must perform the first highlighting of the cursor by
// following this call with a call to <f SCHiLite>.

```

```

status scIMPL_EXPORT      SCCOL_InitialSelect(
                           scColumn*   col,          // @parm <c scColumn>
                           TypeSpec&   spec,          // @parm <t TypeSpec>
                           scSelection*& select );     // @parm <c scSelection>

```

```

// @func Provides information on how good the hit is,
// to be used for selection evaluation.
// the REAL num is the distance squared in micropoints from the
// hitpoint to the nearest charcter and its baseline

```

```

status scIMPL_EXPORT      SCCOL_ClickEvaluate(
                           scColumn*   col,          // @parm <c scColumn>.
                           const scMuPoint& evalpt, // @parm Point to evaluate.
                           REAL&        dist );      // @parm Squared dist in <t MicroPoints>.

```

```

// @func The mouse down click should force a call of SCCOL_StartSelect and
// mouse moves should get <f SCCOL_ExtendSelect>. Effectively the StartSelect
// message forces a flow set to get the focus. Is is an error to call
// ExtendSelect with a column from a different flow set than was called
// from the original StartSelect. Also the client may want to coerce points
// to lie within the column's extents. If the container is rotated the coercion
// should happen in the containers coordinate space to insure correct
// interpretation of the coercion.
// After the first StartSelect message the Selection is accurate so if
// auto scrolling is necessary it may be done, provided the clip region
// is set up correctly.
// <nl>
// <nl>NOTE: The caller should filter out redundant mouse hits, (i.e if the
// current mouse hit is the same as the last mouse hit don't call
// SCCOL_ExtendSelect
//

```

```

status scIMPL_EXPORT      SCCOL_StartSelect(
                           scColumn*   col,          // @parm <c scColumn>

```

```

const scMuPoint& hitpt, // @parm The point in
                    // object coordinates.
HiliteFuncPtr hlfunc, // @parm <t HiliteFuncPtr>
APPDrwCtx drwctx, // @parm <t APPDrwCtx>
scSelection*& sel ); // @parm <c scSelection> to be filled in
                    // by the Composition Toolbox.

```

```

// @func This extends the selection from the scStreamLocation and then may
// be followed with <f SCCOL_ExtendSelect>.

```

```

// @xref <f SCCOL_StartSelect>

```

```

status scIMPL_EXPORT SCCOL_StartSelect(
    scColumn* col, // @parm <c scColumn>
    scStreamLocation& stloc, // @parm <c scStreamLocation>
    const scMuPoint& hitpt, // @parm The hit point in
                            // object coordinates.
    HiliteFuncPtr hlfunc, // @parm <t HiliteFuncPtr>
    APPDrwCtx drwctx, // @parm <t APPDrwCtx>
    scSelection*& sel ); // @parm <c scSelection>

```

```

// @func Extends a selection derived from <f SCCOL_ExtendSelect>. Can
// be used over multiple columns in a flowset.

```

```

status scIMPL_EXPORT SCCOL_ExtendSelect(
    scColumn* col, // @parm <c scColumn>
    const scMuPoint& hitpt, // @parm The hit point in
                            // object coordinates.
    HiliteFuncPtr hlfunc, // @parm <t HiliteFuncPtr>
    APPDrwCtx drwctx, // @parm <t APPDrwCtx>
    scSelection*& sel ); // @parm <c scSelection>

```

```

// @func Converts a <c scSelection> into a mark and point.

```

```

// The mark is guaranteed to logically precede the point.

```

```

// This call is typically used in conjunction with <f SCSEL_Restore> and
// to determine information at the selection point.

```

```

// @xref <f SCCOL_StartSelect>, <f SCCOL_ExtendSelect>, <f SCCOL_InitialSelect>

```

```

status scIMPL_EXPORT SCSEL_Decompose(
    scSelection* sel, // @parm <c scSelection>
    scStreamLocation& mark, // @parm <c scStreamLocation>
    scStreamLocation& point ); // @parm <c scStreamLocation>

```

```

// same as above except that the selection is not sorted

```

```

status scIMPL_EXPORT SCSEL_Decompose2(
    scSelection* sel, // @parm <c scSelection>
    scStreamLocation& mark, // @parm <c scStreamLocation>
    scStreamLocation& point ); // @parm <c scStreamLocation>

```

```

// @func Invalidates the selection in the toolbox, changes the selection
// to null and invalidates the selection in the toolbox

```

```

status scIMPL_EXPORT SCSEL_Invalidate(
    scSelection*& sel ); // @parm <c scSelection>

```

```

/*=====*/

```

```

// @func Sets up a text selection, using the given

```

```

// mark and point. Useful for restoring the selection

```

```

// when re-activating a document, and for undo and redo,

```

```

// especially for undoing arrow and backspace keystrokes.

```

```

// In this call the <c scStreamLocation> need only have the following

```

```

// member variables filled in:

```

```

// <nl> fParaNum

```

```

// <nl> fParaOffset

```

```

// <nl> All the rest are unneeded. After this call subsequent calls

```

```

// to <f SCSEL_Decompose> will fill in the scStreamLocation values

```

```

// correctly.

```

```

status scIMPL_EXPORT      SCSEL_Fire(
                        const scStream*      stream, // @parm <c scStream>
                        const scStreamLocation& mark, // @parm <c scStreamLocation>
                        const scStreamLocation& point, // @parm <c scStreamLocation>
                        scSelection*&        sel, // @parm <c scSelection>
                        Bool                  geometryChange ); // has the layout changed

```

```

/*=====*/
// @func Using a hitpt and a modifier (such as selection of a word or paragraph),
// returns a selection.
//

```

```

status scIMPL_EXPORT      SCCOL_SelectSpecial(
                        scColumn*      col, // @parm <c scColumn>
                        const scMuPoint& hitPt, // @parm Hit point.
                        eSelectModifier mod, // @parm <t eSelectModifier>
                        scSelection*&    sel ); // @parm <c scSelection>

```

```

/*=====*/
// This grows the selection according to the enum eSelectMove found in
// sctypes.h
//

```

```

status scIMPL_EXPORT      SCSEL_Move( scSelection*,
                                      eSelectMove );

```

```

/*=====*/
// This alters the selection point according to the enum eSelectMove found in
// sctypes.h
//

```

```

status scIMPL_EXPORT      SCSEL_Extend( scSelection*, eSelectMove );

```

```

/*=====*/
// select the nth paragraph of a stream - if you go off the end
// status returns noAction
//

```

```

[ ] [ ]
status scIMPL_EXPORT      SCSTR_NthParaSelect( scStream*      streamID,
                                                long           nthPara,
                                                scSelection*    select );

```

```

/*=====*/
// @func Highlights the current selection, using the function pointer passed
// in, the coordinates that will be contained in the call back are in
// object coordinates and MUST be transformed to device coordinates.
//

```

```

status scIMPL_EXPORT      SCSEL_Hilite(
                        scSelection*    sel, // @parm <c scSelection>
                        HiliteFuncPtr   appDrawRect ); // @parm <t HiliteFuncPtr>

```

```

/*===== EDITING MESSAGES =====*/

```

```

// @func Inserts keystrokes into the given stream. Place one or more keystrokes
// into the array. NULL values in the key record array will simply be ignored.
// scKeyRecord* points to an array of keystrokes; numKeys is the number
// of elements in the array.
// Stores information for undo in the key records. Sample code will illustrate
// how to take the inverse values in scKeyRecord and use them to undo the
// insertion of keystrokes.
//

```

```

status scIMPL_EXPORT      SCSEL_InsertKeyRecords(
                        scSelection*    sel, // @parm <c scSelection>
                        short           numRecs, // @parm Number of key recs

```

```

// to follow
scKeyRecord*   keyRecs,    // @parm <c scKeyRecord>
scRedisplList* rInfo );    // @parm <c scRedisplList>
// Redisplay info, arg may be zero.

// @func For immediate redisplay of text that has been altered in editing.
// It redisplay only those lines which the cursor has been on.
// Typing a carriage return should force two lines to be redisplayed
// immediately; likewise for a backspace at the beginning of a line.
// Normally only one line needs to be redisplayed.
// IF THE OPERATION CROSSES COLUMNS, ONLY THE COLUMN
// IN WHICH THE CURSOR ENDS UP IS UPDATED.
//
status scIMPL_EXPORT   SCCOL_UpdateLine(
    scColumn*   col,        // @parm <c scSelection>
    scImmediateRedisp& immred, // @parm <c scImmediateRedisp>
    APPDrwCtx   drwctx );   // @parm <t APPDrwCtx>

// @func Applies a <t TypeSpec> to a <c scSelection>. The TypeSpec replaces
// all specs contained in the selection.
//
status scIMPL_EXPORT   SCSEL_SetTextStyle(
    scSelection* sel,        // @parm <c scSelection>
    TypeSpec      ts,        // @parm <t TypeSpec>
    scRedisplList* rInfo );  // @parm <c scRedisplList>
// Redisplay info, arg may be zero.

//
//
//
//
// Applies a character transformation (to all caps, for example)
// to the selection.
//
// [ ] [ ]
status scIMPL_EXPORT   SCSEL_TextTrans( scSelection*,
    eChTranType,
    int,
    scRedisplList* );

//
// ===== TEXT LEVEL CUT, COPY, PASTE & CLEAR =====/
// These routines cut and paste paragraphs and text characters.
// All relevant paragraph attributes and character attributes are maintained.
// The scrap is maintained in internal Toolbox formats.
//
//
// @func Cuts a selection of text, returning it in the scScrapPtr.
//
status scIMPL_EXPORT   SCSEL_CutText(
    scSelection*   sel,        // @parm <c scSelection>
    scScrapPtr&   scrapPtr,    // @parm <c scScrapPtr>
    scRedisplList* rInfo );    // @parm <c scRedisplList>
// Redisplay info, arg may be zero.

//
// @func Deletes a selection of text.
//
status scIMPL_EXPORT   SCSEL_ClearText(
    scSelection*   sel,        // @parm <c scSelection>
    scRedisplList* rInfo );    // @parm <c scRedisplList>
// Redisplay info, arg may be zero.

```

```

// @func Copies a selection of text, returning the copy in the scScrapPtr.
// <a name="SCSEL_CopyText">-</a>
status scIMPL_EXPORT SCSEL_CopyText(
    scSelection* sel, // @parm <c scSelection>
    scScrapPtr& scrap ); // @parm <c scScrapPtr>

// @func Pastes a selection of text contained in scScrapPtr into a stream
// at the insertion point marked by scSelection. If scSelection is
// a selection of one or more characters, the selected characters
// are deleted from the stream before the scScrapPtr text is pasted in.
// This operation copies the text as it pastes it. Therefore,
// multiple pastes can be made with the same scScrapPtr
// without making any explicit copies.
// @xref <f SCSEL_CutText>, <f SCSEL_CopyText>
status scIMPL_EXPORT SCSEL_PasteText(
    scSelection* sel, // @parm <c scSelection>
    scScrapPtr scrap, // @parm <c scScrapPtr>
    TypeSpec ts, // @parm If NULL uses prev spec.
    scRedisplist* rInfo ); // @parm <c scRedisplist>
// Redisplay info, arg may be zero.

/*=====*/
// @func The following is a useful call to get a stream from a selection and the
// the first typespec in the selection
// @xref SCSEL_GetStream
status scIMPL_EXPORT SCSEL_GetStream(
    const scSelection* sel, // @parm <c scSelection>
    scStream*& str, // @parm <c scStream>
    TypeSpec& ts ); // @parm The first <t TypeSpec>.

/*=====*/
// see doc in html file
// @xref SCSEL_InsertField
status SCSEL_InsertField( scSelection*,
    const clField&,
    TypeSpec&,
    scRedisplist* damage );

/*=====*/
// The following are for spell checkings */

// actual definition in SCTYPES.H
// Substitution function
// inWord and outWord are null terminated strings
// outWord is allocated and freed by application
//
// the substitution should function returns
// - successful if the word has been changed
// - noAction if no change is necessary
// - other error to be propogated back to app
//
// typedef status (*SubstituteFunc)( UCS2**outWord, UCS2*inWord );
//
//
/*=====*/
// DEPRECATED
// check spelling in selection,
// this is best used in conjunction with SCMoveSelect
//
// [ ] [ ]
status scIMPL_EXPORT SCSEL_Iter( scSelection* selectID,

```

```
SubstituteFunc func,
scRedisplList* damage );
```

```
/*=====*/
```

```
// DEPRECATED
```

```
// check spelling in the stream
```

```
//
```

```
// [ ] [ ]
```

```
status scIMPL_EXPORT SCSTR_Iter( scStream* streamID,
                                SubstituteFunc func,
                                scRedisplList* damage );
```

```
/*=====*/
```

```
// DEPRECATED
```

```
// [ ] [ ]
```

```
status scIMPL_EXPORT SCSTR_Search( scStream* streamID,
                                   const UCS2* findString,
                                   SubstituteFunc func,
                                   scRedisplList* damage );
```

```
// @func Search for the string from the current selection. When
```

```
// the string is found move the selection to it.
```

```
status scIMPL_EXPORT SCSEL_FindString(
    scSelection* select, // @parm <c scSelection>
    const UCS2* findString ); // @parm <t UCS2> string to find.
```

```
/*=====*/
```

```
// this returns whether or not the column potentially has text, if
```

```
// because of reformatting nothing lands in here we will still return
```

```
// true successful == has text
```

```
//
```

```
//
```

```
//
```

```
// [ ] [ ]
```

```
status scIMPL_EXPORT SCCOL_HasText( scColumn* colID );
```

```
//
```

```
/*=====*/
```

```
// at start up this will have the first token selected
```

```
//
```

```
class stTokenIter {
```

```
public:
```

```
    virtual void release() = 0;
```

```
    virtual void reset() = 0;
```

```
//
```

```
    virtual int // sets a selection of the iterators para
                paraselection( scSelection* ) = 0;
```

```
//
```

```
    virtual int // sets selection for the token iterator
                setselection( scSelection* ) = 0;
```

```
//
```

```
    // retrieves the next token,
    // the string should have its size set in the
    // len field, if the token will not fit
    // in the string, a negative number is returned
    // that specifies the size, try again with a sufficient
    // size string
```

```
    virtual int gettoken( stUnivString& ) = 0;
```

```
//
```

```
    virtual int // replaces the current token
                replacetoken( stUnivString& ) = 0;
```

```
//
```

```
    virtual int // moves to next token
                next() = 0;
```

```
};
```

```
class stContUnitIter {
```

```
public:
```

```
    virtual void release() = 0;
```

```
    virtual void reset() = 0;
```

```
    virtual int gettokeniter( stTokenIter*& ) = 0;
```

```
    virtual int next() = 0;
```


$$\begin{array}{ccccccc} \{1^{\alpha_1}\} & \{1^{\alpha_2}\} & \{1^{\alpha_3}\} & \{1^{\alpha_4}\} & \{1^{\alpha_5}\} & \{1^{\alpha_6}\} & \{1^{\alpha_7}\} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \{1^{\alpha_{n-1}}\} & \{1^{\alpha_n}\} & \{1^{\alpha_{n+1}}\} & \{1^{\alpha_{n+2}}\} & \{1^{\alpha_{n+3}}\} & \{1^{\alpha_{n+4}}\} & \{1^{\alpha_{n+5}}\} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \{1^{\alpha_{n+1}}\} & \{1^{\alpha_{n+2}}\} & \{1^{\alpha_{n+3}}\} & \{1^{\alpha_{n+4}}\} & \{1^{\alpha_{n+5}}\} & \{1^{\alpha_{n+6}}\} & \{1^{\alpha_{n+7}}\} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \{1^{\alpha_{n+1}}\} & \{1^{\alpha_{n+2}}\} & \{1^{\alpha_{n+3}}\} & \{1^{\alpha_{n+4}}\} & \{1^{\alpha_{n+5}}\} & \{1^{\alpha_{n+6}}\} & \{1^{\alpha_{n+7}}\} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \{1^{\alpha_{n+1}}\} & \{1^{\alpha_{n+2}}\} & \{1^{\alpha_{n+3}}\} & \{1^{\alpha_{n+4}}\} & \{1^{\alpha_{n+5}}\} & \{1^{\alpha_{n+6}}\} & \{1^{\alpha_{n+7}}\} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \{1^{\alpha_{n+1}}\} & \{1^{\alpha_{n+2}}\} & \{1^{\alpha_{n+3}}\} & \{1^{\alpha_{n+4}}\} & \{1^{\alpha_{n+5}}\} & \{1^{\alpha_{n+6}}\} & \{1^{\alpha_{n+7}}\} \end{array}$$

File: SCAPPTEX.H

\$Header: /Projects/Toolbox/ct/SCAPPTEX.H 2 5/30/97 8:45a Wmanis \$

Contains: The class for passing content plus typo state
back between client and toolbox.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

@doc

```

*****/

#ifdef _H_SCAPPTEX
#define _H_SCAPPTEX

#ifdef SCMACINTOSH
#pragma once
#endif

#include "sctypes.h"

class stTextImportExport {
public:
    static stTextImportExport& MakeTextImportExport( int encoding = 1 );

    virtual void    release() = 0;

    // writing
    virtual void    StartPara( TypeSpec& ) = 0;
    virtual void    SetParaSpec( TypeSpec& ) = 0;
    virtual void    PutString( const uchar*, int, TypeSpec& ) = 0;
    virtual void    PutString( stUnivString&, TypeSpec& ) = 0;
    virtual void    PutChar( UCS2, TypeSpec& ) = 0;

    // reading
    virtual int     NextPara( TypeSpec& ) = 0;
    virtual int     GetChar( UCS2&, TypeSpec& ) = 0;
    virtual void    reset() = 0;
    virtual void    resetpara() = 0;
};

#endif /* _H_SCAPPTEX */

```

File: SCAPPTEX.C

\$Header: /Projects/Toolbox/ct/SCAPPTEX.CPP 2 5/30/97 8:45a Wmanis \$

Contains:

This module takes a handle to APPTexRun structure and manages the memory (specs & chars) associated with it. Locking and unlocking. It also maintains some internal structures that used for reading the text between the lock and unlock calls. Refer to the APPTexRun in SCTextExch.h.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

```

*****/
#include "scapptex.h"
#include "scparagr.h"
/* ===== */
stPara::stPara( ) :
    paraspec_( 0 )
{
    reset();
}
/* ===== */
stPara::stPara( TypeSpec& pspec ) :
    paraspec_( pspec )
{
    if ( pspec.ptr() )
        specs_.AppendSpec( pspec, 0 );
    reset();
}
/* ===== */
stPara::~stPara()
{
}
/* ===== */
void stPara::append( TypeSpec& ts )
{
    if ( ts != specs_.SpecAtOffset( ch_.NumItems() ) )
        specs_.AppendSpec( ts, ch_.NumItems() );
}
/* ===== */
void stPara::append( const uchar* ch, int len )
{
    for ( int i = 0; i < len; i++ )
        ch_.Append( (UCS2)ch[i] );
}
/* ===== */

```

```

void stPara::append( stUnivStri ustr )
{
    for ( unsigned i = 0; i < ustr.len; i++ )
        ch_.Append( (UCS2)ustr.ptr[i] );
}

/* ===== */

void stPara::append( UCS2 ch )
{
    ch_.Append( ch );
}

/* ===== */

stPara& stPara::operator=( const stPara& p )
{
    ch_.RemoveAll();
    for ( int i = 0; i < p.ch_.NumItems(); i++ )
        ch_.Append( p.ch_[i] );

    choffset_ = p.choffset_;

    specs_.RemoveAll();
    for ( i = 0; i < p.specs_.NumItems(); i++ )
        specs_.Append( p.specs_[i] );

    paraspec_ = p.paraspec_;
    return *this;
}

/* ===== */

int stPara::get( UCS2& ch, TypeSpec& spec )
{
    if ( choffset_ < ch_.NumItems() ) {
        spec = specs_.SpecAtOffset( choffset_ );
        ch = ch_[choffset_++];
        return choffset_;
    }
    return 0;
}

/* ===== */

int stPara::validate() const
{
    scAssert( paraspec_.ptr() );
    specs_.DebugRun( "stPara::validate" );
    return 1;
}

/* ===== */

void stPara::setparaspec( TypeSpec& ts )
{
    paraspec_ = ts;
    scAssert( ch_.NumItems() == 0 );
    specs_.AppendSpec( ts, 0 );
}

/* ===== */

int stPara::complete()
{
    if ( specs_.NumItems() == 1 )
        specs_.AppendSpec( paraspec_, 0 );

    return validate();
}

/* ===== */

```

```

class stTIEImp : public stTextImportExport {
public:
    stTIEImp();
    ~stTIEImp();

    void release();

    // writing
    virtual void StartPara( TypeSpec& );
    virtual void SetParaSpec( TypeSpec& );
    virtual void PutString( const uchar*, int, TypeSpec& );
    virtual void PutString( stUnivString&, TypeSpec& );
    virtual void PutChar( UCS2, TypeSpec& );

    // reading
    virtual int NextPara( TypeSpec& );
    virtual int GetChar( UCS2&, TypeSpec& );
    virtual void reset();
    virtual void resetpara();

protected:
    stPara& currentPara();

    int32 pindex_;
    scSizeableArrayD<stPara> paras_;
};

/* ===== */
stTextImportExport& stTextImportExport::MakeTextImportExport( int encoding )
{
    stTIEImp* stimp = new stTIEImp();
    return *stimp;
}

/* ===== */
stTIEImp::stTIEImp()
{
    reset();
}

/* ===== */
stTIEImp::~stTIEImp()
{
}

/* ===== */
void stTIEImp::release()
{
    delete this;
}

/* ===== */
stPara& stTIEImp::currentPara()
{
    return paras_[pindex_];
}

/* ===== */
// importing

void stTIEImp::StartPara( TypeSpec& ts )
{
    if ( paras_.NumItems() > 0 )
        paras_[pindex_].complete();

    stPara newPara( ts );
    paras_.Append( newPara );
}

```

```

    pindex_ = paras_.NumItems()-1;
}

/* ===== */

void stTIEImp::SetParaSpec( TypeSpec& ts )
{
    stPara& p = currentPara();
    p.setparaspec( ts );
}

/* ===== */

void stTIEImp::PutString( const uchar* str, int len, TypeSpec& ts )
{
    stPara& p = currentPara();
    p.append( ts );
    p.append( str, len );
}

/* ===== */

void stTIEImp::PutString( stUnivString& ustr, TypeSpec& ts )
{
    stPara& p = currentPara();
    p.append( ts );
    p.append( ustr );
}

/* ===== */

void stTIEImp::PutChar( UCS2 ch, TypeSpec& ts )
{
    stPara& p = currentPara();
    p.append( ts );
    p.append( ch );
}

/* ===== */
// reading
int stTIEImp::NextPara( TypeSpec& ts )
{
    pindex_++;
    if ( pindex_ < paras_.NumItems() ) {
        ts = paras_[pindex_].paraspec();
        paras_[pindex_].validate();
        return pindex_;
    }
    return -1;
}

/* ===== */

int stTIEImp::GetChar( UCS2& ch, TypeSpec& ts )
{
    stPara& p = currentPara();
    return p.get( ch, ts );
}

/* ===== */

void stTIEImp::reset()
{
    pindex_ = -1;
}

/* ===== */

void stTIEImp::resetpara()
{
    stPara& p = currentPara();
    p.reset();
}

```

}

[illegible]

File: SCAPPTYP.H

\$Header: /Projects/Toolbox/ct/SCAPPTYP.H 2 5/30/97 8:45a Wmanis \$

Contains: Defintion by client of data types.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

Use this to define application types for proper type checking, these are types that are by and large passed thru the Composition Toolbox and thus type information is superfluous

@doc

*****/

```
#ifndef _H_SCAPPTYP
#define _H_SCAPPTYP
```

```
#ifdef _WINDOWS
#include <windows.h>
#endif
```

```

===== */
===== */
```

```
class DemoView;
class CAGText;
class ApplIOContext;
```

```
enum {
    Japanese      = 0,
    English,
    Spanish,
    Italian,
    Portuguese,
    French,
    German,
    Dutch,
    Bokmal,
    Nynorsk,
    Swedish,
    Danish,
    Icelandic,
    Greek,
    Turkish,
    Russian,
    Croatian,
    Finnish,
    Miscellaneous,
    MAX_LANGUAGES
};
```

```
// @type APPLanguage | An abstract type/magic cookie that the Composition Toolbox
// may use to specify hyphenation language.
typedef short          APPLanguage;
```

```
// @type APPColor | An abstract type/magic cookie that the Composition Toolbox
// may use to specify color.
```

```

//
typedef COLORREF                APPColor;        /* color reference */

// @type APPDrwCtx | An abstract type/magic cookie that the Composition Toolbox
// may use to pass thru drawing contexts.
//
typedef CAGText*                APPDrwCtx;        // drawing context

// @type APPFont | An abstract type/magic cookie that the Composition Toolbox
// may use to retrieve and specify font information.
//
typedef const scChar*           APPFont;

// @type APPRender | An abstract type/magic cookie that the Composition Toolbox
// may use to specify font information plus additional drawing attributes
// that the client may wish to use (e.g. drop shadow ). Typically used when
// the traditional values returned by the font sub-system in Quickdraw or
// GDI would not suffice.
// @xref <t APPFont>
typedef struct RenderDef*       APPRender;

// @type TypeSpec | An abstract type/magic cookie that the Composition Toolbox
// may use to retrieve typographic state information.
//
#include "refcnt.h"
class stSpec : public RefCount
{
};
typedef class RefCountPtr<stSpec>    TypeSpec;
typedef class RefCountPtr<stSpec>    scFontRender;

// @type APPColumn | An abstract type/magic cookie to be filled in
// appropriately by the client.
//
typedef CAGText*                APPColumn;

// @type APPCtxPtr | An abstract type/magic cookie for use in file i/o.
// @xref <t IOFuncPtr>
typedef CAGText*                APPCtxPtr;

/* ===== */
#endif

```

File: SCARRAY.CPP

\$Header: /btoolbox/lib/SCARRAY.CPP 3 3/31/96 3:48p Will \$

Contains: Templates for Vector

Written by: Manis

Copyright (c) 1989-95 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

*****/

```
#ifdef DEFINE_TEMPLATES
```

```
#if defined( __MWERKS__ )
```

```
#include <stdlib.h>
```

```
#else
```

```
#include <malloc.h>
```

```
#endif
```

```
#include <string.h>
```

```
#include <assert.h>
```

```
inline
```

```
void *operator new(size_t, void *p)
```

```
{
    return p;
}
```

```
#ifndef scAssert
```

```
#define stAssert    assert
```

```
#else
```

```
#define stAssert    scAssert
```

```
#endif
```

```
#define stMAX( a, b )      ((a)>(b)?(a):(b))
```

```
#define stMIN( a, b )      ((a)<(b)?(a):(b))
```

```
template <class T>
```

```
scStaticArray<T>::scStaticArray( int num, T* mem ) :
```

```
    scArray<T>( num, mem ? mem : new T[num] )
```

```
{
}
```

```
template <class T>
```

```
scStaticArray<T>::~~scStaticArray()
```

```
{
    delete [] items_;
}
```

```
#define kInitBlockSize 4
```

```
template <class T>
```

```
scSizeableArray<T>::scSizeableArray() :
```

```
    elemSlots_( 4 ),
```

```
    blockSize_( kInitBlockSize ),
```

```
    scArray<T>( 0, (T*)malloc( kInitBlockSize * sizeof( T ) ) )
```

```
{
    ClearMem( 0 );
```

}

```
template <class T>
scSizeableArray<T>::~scSizeableArray()
{
    if ( items_ )
        free( items_ );
}
```

```
template <class T>
void scSizeableArray<T>::Remove( int index )
{
    stAssert( index < numItems_ );

    memmove( items_ +index, items_ +index + 1,
        ( numItems_ - index - 1 ) * sizeof( T ) );

    numItems_ -= 1;

    ShrinkSlots();
}
```

```
template <class T>
void scSizeableArray<T>::RemoveAll()
{
    numItems_ = 0;
    ShrinkSlots();
}
```

```
template <class T>
void scSizeableArray<T>::SetNumSlots( int numSlots )
{
    int numBlocks = stMAX( 1, numSlots / blockSize_ + ( numSlots % blockSize_ ? 1 : 0 ) );
    SizeSlots( numBlocks * blockSize_ );
}
```

```
template <class T>
int scSizeableArray<T>::Append( const T& elem )
{
    SetNumSlots( numItems_ + 1 );
    items_[numItems_] = elem;
    return numItems_++;
}
```

```
template <class T>
T& scSizeableArray<T>::Grow()
{
    SetNumSlots( numItems_ + 1 );
    return items_[numItems_++];
}
```

```
template <class T>
void scSizeableArray<T>::Insert( int index, const T& elem )
{
    SetNumSlots( numItems_ + 1 );

    if ( numItems_ - index > 0 )
        memmove( items_ + index + 1, items_ + index,
            ( numItems_ - index ) * sizeof(T) );

    items_[index] = elem;
    numItems_++;
}
```

```
template <class T>
void scSizeableArray<T>::Set( int index, const T& elem )
{
    if ( index >= numItems_ ) {
        SetNumSlots( index + 1 );
        numItems_ = index + 1;
    }
}
```

```

    items_[index] = elem;
}

```

```

template <class T>
void scSizeableArray<T>::SizeSlots( unsigned numItems )
{
    // do not shrink if we are retaining memory or if no resizing is
    // necessary
    //
    if ( elemSlots_ == numItems || ( numItems < elemSlots_ && retainMem_ ) )
        return;

    stAssert( numItems >= blockSize_ );

    long oldSize = elemSlots_;

    if ( items_ == 0 )
        throw( -1 );

    items_ = (T*)realloc( items_, sizeof(T) * numItems );
    elemSlots_ = numItems;
    ClearMem( oldSize );
}

```

```

template <class T>
void scSizeableArray<T>::GrowSlots( int newItems )
{
    int oldSize = elemSlots_;

    if ( items_ == 0 )
        throw( -1 );

    items_ = (T*)realloc( items_, sizeof(T) * ( elemSlots_ + newItems ) );
    elemSlots_ += newItems;
    ClearMem( oldSize );
}

```

```

template <class T>
void scSizeableArray<T>::ClearMem( unsigned oldsize )
{
    // either we do need to clear memory or we have shrunk it
    if ( oldsize >= elemSlots_ )
        return;

    for ( unsigned index = oldsize; index < elemSlots_; index++ )
        memset( items_ + index, 0, sizeof(T) );
}

```

```

#if 0
template <class T>
void scSizeableArray<T>::exch( const scSizeableArray<T>& arr )
{
    scArray<T>::operator=( arr );

    scSizeableArray<T>& array = (scSizeableArray<T>&)arr;

    unsigned tmp = elemSlots_;
    elemSlots_ = array.elemSlots_;
    array.elemSlots_ = tmp;

    tmp = blockSize_;
    blockSize_ = array.blockSize_;
    array.blockSize_ = blockSize_;

    tmp = retainMem_;
    retainMem_ = array.retainMem_;
    array.retainMem_ = retainMem_;
}
#endif

```

```

template<class T, class CT>
int scBinarySortedArray<T,CT>::Find( const T& val, int* insertIndexP ) const
{
    int    low      = 0;
    int    high     = numItems_ - 1;
    int insertIndex = 0;

    while ( low <= high ) {
        int index = (low + high) / 2;
        int found = CT::Compare( val, items_[index] );
        if ( found == 0 )
            return index;
        else if ( found < 0 )
            high = index - 1, insertIndex = index;
        else
            low = index + 1, insertIndex = index + 1;
    }

    insertIndexP ? *insertIndexP = insertIndex : 0;
    return -1;
}

template<class T, class CT>
int scBinarySortedArray<T,CT>::Find1( const T& val, const CT& ct, int* insertIndexP ) const
{
    int    low      = 0;
    int    high     = numItems_ - 1;
    int insertIndex = 0;

    while ( low <= high ) {
        int index = (low + high) / 2;
        int found = ct.Compare1( val, items_[index] );
        if ( found == 0 )
            return index;
        else if ( found < 0 )
            high = index - 1, insertIndex = index;
        else
            low = index + 1, insertIndex = index + 1;
    }

    insertIndexP ? *insertIndexP = insertIndex : 0;
    return -1;
}

template<class T, class CT>
int scBinarySortedArray<T,CT>::SortInsert( const T& item )
{
    int    index;
    if ( Find( item, &index ) < 0 )
        Insert( index, item );
    return index;
}

////

template <class T>
scStaticArrayD<T>::scStaticArrayD( int num, T* mem ) :
    scArray<T>( num, mem ? mem : new T[num] )
{
}

template <class T>
scStaticArrayD<T>::~~scStaticArrayD()
{
    for ( int i = 0; i < numItems_; i++) {
#ifdef __WATCOMC__
        items_[i].~T();
#elif defined(_MAC)
        T& tp = items_[i];
        tp.~T();
#else
        items_[i].T::~~T();
#endif
    }
}

```

```

    }
    delete [] items_;
}

#define kInitBlockSize 4

template <class T>
scSizeableArrayD<T>::scSizeableArrayD() :
    elemSlots_( 4 ),
    blockSize_( kInitBlockSize ),
    scArray<T>( 0, (T*)malloc( kInitBlockSize * sizeof( T ) ) )
{
    ClearMem( 0 );
}

template <class T>
scSizeableArrayD<T>::~scSizeableArrayD()
{
    for ( unsigned i = 0; i < elemSlots_; i++)
        deleteItem( i );

    if ( items_ )
        free( items_ );
}

template <class T>
void scSizeableArrayD<T>::constructItem( int index )
{
    (void)new( items_ + index ) T;
}

template <class T>
void scSizeableArrayD<T>::deleteItem( int index )
{
#ifdef __WATCOMC__
    items_[index].~T();
#elif defined(_MAC)
    T& tp = items_[index];
    tp.~T();
#else
    items_[index].T::~~T();
#endif
}

template <class T>
void scSizeableArrayD<T>::Remove( int index )
{
    stAssert( index < numItems_ );
    deleteItem( index );

    if ( numItems_ - index - 1 )
        memmove( items_ + index, items_ + index + 1,
            ( numItems_ - index - 1 ) * sizeof( T ) );

    numItems_ -= 1;

    ShrinkSlots();
}

template <class T>
void scSizeableArrayD<T>::RemoveAll()
{
    for ( int i = 0; i < numItems_; i++)
        deleteItem( i );

    numItems_ = 0;
    ShrinkSlots();
}

template <class T>
void scSizeableArrayD<T>::SetNumSlots( int numSlots )
{
    int numBlocks = stMAX( 1, numSlots / blockSize_ + ( numSlots % blockSize_ ? 1 : 0 ) );

```

```

    SizeSlots( numBlocks * blockSize_ );
}

template <class T>
int scSizeableArrayD<T>::Append( const T& elem )
{
    SetNumSlots( numItems_ + 1 );
    items_[numItems_] = elem;
    return numItems_++;
}

template <class T>
T& scSizeableArrayD<T>::Grow()
{
    SetNumSlots( numItems_ + 1 );
    return items_[numItems_++];
}

template <class T>
void scSizeableArrayD<T>::Insert( int index, const T& elem )
{
    SetNumSlots( numItems_ + 1 );

    if ( numItems_ - index > 0 )
        memmove( items_ + index + 1, items_ + index,
            ( numItems_ - index ) * sizeof(T) );

    constructItem( index );
    items_[index] = elem;
    numItems_++;
}

template <class T>
void scSizeableArrayD<T>::Set( int index, const T& elem )
{
    if ( index >= numItems_ ) {
        SetNumSlots( index + 1 );
        numItems_ = index + 1;
    }

    items_[index] = elem;
}

template <class T>
void scSizeableArrayD<T>::SizeSlots( unsigned numItems )
{
    // do not shrink if we are retaining memory or if no resizing is
    // necessary
    //
    if ( elemSlots_ == numItems || ( numItems < elemSlots_ && retainMem_ ) )
        return;

    stAssert( numItems >= blockSize_ );

    long oldSize = elemSlots_;

    if ( items_ == 0 )
        throw( -1 );

    items_ = (T*)realloc( items_, sizeof(T) * numItems );
    elemSlots_ = numItems;
    ClearMem( oldSize );
}

template <class T>
void scSizeableArrayD<T>::GrowSlots( int newItems )
{
    int oldSize = elemSlots_;

```

```

    if ( items_ == 0 )
        throw( -1 );

    items_ = (T*)realloc( items_, sizeof(T) * ( elemSlots_ + newItems ) );
    elemSlots_ += newItems;
    ClearMem( oldSize );
}

```

```

template <class T>
void scSizeableArrayD<T>::ClearMem( unsigned oldsize )
{
    // either we do need to clear memory or we have shrunk it
    if ( oldsize >= elemSlots_ )
        return;

    for ( unsigned index = oldsize; index < elemSlots_; index++ )
        constructItem( index );
}

```

```

#ifdef 0
template <class T>
void scSizeableArrayD<T>::exch( const scSizeableArrayD<T>& arr )
{
    scArray<T>::operator=( arr );

```

```

    scSizeableArrayD<T>& array = (scSizeableArrayD<T>&)arr;

```

```

    unsigned tmp = elemSlots_;
    elemSlots_ = array.elemSlots_;
    array.elemSlots_ = tmp;

```

```

    tmp = blockSize_;
    blockSize_ = array.blockSize_;
    array.blockSize_ = blockSize_;

```

```

    tmp = retainMem_;
    retainMem_ = array.retainMem_;
    array.retainMem_ = retainMem_;

```

```

#endif

```

```

template<class T, class CT>
int scBinarySortedArrayD<T,CT>::Find( const T& val, int* insertIndexP ) const

```

```

{
    int low = 0;
    int high = numItems_ - 1;
    int insertIndex = 0;

    while ( low <= high ) {
        int index = (low + high) / 2;
        int found = CT::Compare( val, items_[index] );
        if ( found == 0 )
            return index;
        else if ( found < 0 )
            high = index - 1, insertIndex = index;
        else
            low = index + 1, insertIndex = index + 1;
    }

```

```

    insertIndexP ? *insertIndexP = insertIndex : 0;
    return -1;
}

```

```

template<class T, class CT>
int scBinarySortedArrayD<T,CT>::Find1( const T& val, const CT& ct, int* insertIndexP ) const
{

```

```

    int low = 0;
    int high = numItems_ - 1;
    int insertIndex = 0;

```

```

    while ( low <= high ) {
        int index = (low + high) / 2;

```


File: SCARRAY.H

\$Header: /btoolbox/lib/SCARRAY.H 3 3/31/96 3:48p Will \$

Contains: Templates for Vector

Written by: Manis

Copyright (c) 1989-95 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/

#ifndef _H_SCARRAY
#define _H_SCARRAY

#ifdef _DEBUG
#define ifdebug( x )    x
#else
#define ifdebug( x )
#endif

/* ===== */

template <class T>
class scAutoDelete {
public:
    scAutoDelete( T* ptr = 0 ): ptr_( ptr ) {}
    ~scAutoDelete() { freePtr(); }

    T*   deref() const { return ptr_; }
    T*   operator->() const { return ptr_; }
    T&   operator*() const { return *ptr_; }

    void operator=( T* p ) { if ( ptr_ ) freePtr(); ptr_ = p; }
    int  operator==( const scAutoDelete<T>& p )
        { return ptr_ == p.deref(); }

private:
    void freePtr() { delete ptr_, ptr_ = 0; }
    T*   ptr_;
};

/* ===== */

template <class T>
class scArray {
public:
    int      NumItems( void ) const;

    T&       operator[] ( int n );
    const T& operator[] ( int n ) const;

    T*       ptr( void );
    const T* ptr( void ) const;

    // void exch( const scArray<T>& );
protected:
    scArray() :
        numItems_(0),
        items_(0){}

```

```

        scArray( int num, T* mem ) :
            numItems_( num ),
            items_( mem ) {}

    int      numItems_;
    T*       items_;

#ifdef __MWERKS__
        // the following are not declared because of the
        // deep vs shallow copy problem
        scArray( const scArray& );    // not declared
        scArray& operator=( const scArray& );    // not declared
#endif
};

template <class T>
#ifdef __MWERKS__
inline
#endif
int scArray<T>::NumItems() const
{
    return numItems_;
}

template <class T>
#ifdef __MWERKS__
inline
#endif
T& scArray<T>::operator[] ( int n )
{
    return items_[ n ];
}

template <class T>
#ifdef __MWERKS__
inline
#endif
const T& scArray<T>::operator[] ( int n ) const
{
    return items_[ n ];
}

template <class T>
#ifdef __MWERKS__
inline
#endif
T* scArray<T>::ptr()
{
    return items_;
}

template <class T>
#ifdef __MWERKS__
inline
#endif
const T* scArray<T>::ptr() const
{
    return items_;
}

#if 0
template <class T>
#ifdef __MWERKS__
inline
#endif
void scArray<T>::exch( const scArray<T>& arr )

```

```

{
    scArray<T> array = (scArray<T>&)arr;

    int t = array.numItems_;
    array.numItems_ = numItems_;
    numItems_ = array.numItems_;

    T* tmp = array.items_;
    array.items_ = items_;
    items_ = tmp;
}
#endif

/* ----- */

template <class T>
class scStaticArray : public scArray<T> {
public:
    scStaticArray( int num, T* mem = 0 );
    ~scStaticArray();
};

/* ----- */

template <class T>
class scStaticArrayD : public scArray<T> {
public:
    scStaticArrayD( int num, T* mem = 0 );
    ~scStaticArrayD();
};

/* ----- */

template <class T>
class scSizeableArray : public scArray<T> {
public:
    scSizeableArray();
    ~scSizeableArray();

    void Remove( int );
    void RemoveAll( void );
    int Append( const T& );

    T& Grow();

    void Insert( int, const T& );
    void Set( int, const T& );
    void SetNumSlots( int numSlots );
    void SetRetainMem( int tf )
    {
        retainMem_ = tf ? 1 : 0;
    }
    int GetRetainMem( void ) const
    {
        return retainMem_;
    }

// void exch( const scSizeableArray<T>& );

private:
    void MoreSlots( void )
    {
        GrowSlots( blockSize_ );
    }
    void ShrinkSlots( void )
    {
        SetNumSlots( numItems_ );
    }

    void SizeSlots( unsigned );

```

```

void      GrowSlots( int );
void      ClearMem( unsigned );

unsigned   elemSlots_;      // num of elements potentially in allocated space
unsigned   blockSize_ : 16; // for growing and shrinking we grow in greater
                             // than one element unit - this is that unit
                             // typically 4
unsigned   retainMem_ : 1;  // do not shrink memory if this is set
};

```

```
/* ===== */
```

```

template <class T>
class scSizeableArrayD : public scArray<T> {
public:
    scSizeableArrayD();
    ~scSizeableArrayD();

    void      Remove( int );
    void      RemoveAll( void );

    T&        Grow();

    int        Append( const T& );
    void      Insert( int, const T& );

    void      Set( int, const T& );

    void      SetNumSlots( int numSlots );

    void      SetRetainMem( int tf )
    {
        retainMem_ = tf ? 1 : 0;
    }

    int        GetRetainMem( void ) const
    {
        return retainMem_;
    }

    void      exch( const scSizeableArrayD<T>& );

private:
    void      constructItem( int );
    void      deleteItem( int );

    void      MoreSlots( void )
    {
        GrowSlots( blockSize_ );
    }

    void      ShrinkSlots( void )
    {
        SetNumSlots( numItems_ );
    }

    void      SizeSlots( unsigned );
    void      GrowSlots( int );
    void      ClearMem( unsigned );

    unsigned   elemSlots_;      // num of elements potentially in allocated space
    unsigned   blockSize_ : 16; // for growing and shrinking we grow in greater
                             // than one element unit - this is that unit
                             // typically 4
    unsigned   retainMem_ : 1;  // do not shrink memory if this is set
};

```

```
/* ===== */
```

```

// CT is a comparison class - it must have a method
// Compare( const T&, const T& )

```

```
template <class T, class CT>
```

[illegible]

File: SCBEZBLE.C

\$Header: /Projects/Toolbox/ct/SCBEZBLE.CPP 2 5/30/97 8:45a Wmanis \$

Contains: the blending values for computing beziers

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

*****/

/* this contains standard function values, we use tables instead of actually
* computing the value
*/

#include "scbezier.h"

scBezBlendValue bezblend[scBezBlendSize] = {

/* this one appears in all */

scBezFactor(0x0000, 1.0000000), /* (0 0) */
scBezFactor(0x0000, 0.0000000), /* (0 1) */
scBezFactor(0x0000, 0.0000000), /* (0 2) */
scBezFactor(0x0000, 0.0000000), /* (0 3) */

#ifdef SubDiv256

scBezFactor(0xfd02, 0.9883270), /* (1 0) */
scBezFactor(0x02fa, 0.0116274), /* (1 1) */
scBezFactor(0x0002, 0.0000456), /* (1 2) */
scBezFactor(0x0000, 0.0000001), /* (1 3) */

#endif

#ifdef SubDiv128

{
scBezFactor(0xfa0b, 0.9767451), /* (2 0) */
scBezFactor(0x05e8, 0.0230727), /* (2 1) */
scBezFactor(0x000b, 0.0001817), /* (2 2) */
scBezFactor(0x0000, 0.0000005), /* (2 3) */
}

#endif

#ifdef SubDiv256

{
scBezFactor(0xf71a, 0.9652541), /* (3 0) */
scBezFactor(0x08ca, 0.0343371), /* (3 1) */
scBezFactor(0x001a, 0.0004072), /* (3 2) */
scBezFactor(0x0000, 0.0000016), /* (3 3) */
}

#endif

#ifdef SubDiv64

{
scBezFactor(0xf42f, 0.9538536), /* (4 0) */
scBezFactor(0x0ba0, 0.0454216), /* (4 1) */
scBezFactor(0x002f, 0.0007210), /* (4 2) */
}

```

    scBezFactor( 0x0000,      0.000038 ) /* (4 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xf14a,      0.9425432 ), /* (5 0) */
    scBezFactor( 0x0e6b,      0.0563273 ), /* (5 1) */
    scBezFactor( 0x0049,      0.0011221 ), /* (5 2) */
    scBezFactor( 0x0000,      0.0000075 ), /* (5 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0xee6b,      0.9313226 ), /* (6 0) */
    scBezFactor( 0x112a,      0.0670552 ), /* (6 1) */
    scBezFactor( 0x0069,      0.0016093 ), /* (6 2) */
    scBezFactor( 0x0000,      0.0000129 ), /* (6 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xeb91,      0.9201913 ), /* (7 0) */
    scBezFactor( 0x13de,      0.0776065 ), /* (7 1) */
    scBezFactor( 0x008e,      0.0021817 ), /* (7 2) */
    scBezFactor( 0x0001,      0.0000204 ), /* (7 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0xe8be,      0.9091492 ), /* (8 0) */
    scBezFactor( 0x1686,      0.0879822 ), /* (8 1) */
    scBezFactor( 0x00ba,      0.0028381 ), /* (8 2) */
    scBezFactor( 0x0002,      0.0000305 ), /* (8 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xe5f0,      0.8981957 ), /* (9 0) */
    scBezFactor( 0x1922,      0.0981833 ), /* (9 1) */
    scBezFactor( 0x00ea,      0.0035775 ), /* (9 2) */
    scBezFactor( 0x0002,      0.0000435 ), /* (9 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0xe328,      0.8873305 ), /* (10 0) */
    scBezFactor( 0x1bb3,      0.1082110 ), /* (10 1) */
    scBezFactor( 0x0120,      0.0043988 ), /* (10 2) */
    scBezFactor( 0x0003,      0.0000596 ), /* (10 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xe065,      0.8765534 ), /* (11 0) */
    scBezFactor( 0x1e39,      0.1180664 ), /* (11 1) */
    scBezFactor( 0x015b,      0.0053009 ), /* (11 2) */
    scBezFactor( 0x0005,      0.0000793 ), /* (11 3) */
},
#endif

```

```
#ifdef SubDiv64
{
    scBezFactor( 0xdda9,      0.8658638 ), /* (12 0) */
    scBezFactor( 0x20b4,      0.1277504 ), /* (12 1) */
    scBezFactor( 0x019b,      0.0062828 ), /* (12 2) */
    scBezFactor( 0x0006,      0.0001030 ), /* (12 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xdaf2,      0.8552615 ), /* (13 0) */
    scBezFactor( 0x2323,      0.1372642 ), /* (13 1) */
    scBezFactor( 0x01e1,      0.0073434 ), /* (13 2) */
    scBezFactor( 0x0008,      0.0001310 ), /* (13 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0xd841,      0.8447461 ), /* (14 0) */
    scBezFactor( 0x2588,      0.1466088 ), /* (14 1) */
    scBezFactor( 0x022b,      0.0084815 ), /* (14 2) */
    scBezFactor( 0x000a,      0.0001636 ), /* (14 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xd595,      0.8343173 ), /* (15 0) */
    scBezFactor( 0x27e1,      0.1557854 ), /* (15 1) */
    scBezFactor( 0x027b,      0.0096962 ), /* (15 2) */
    scBezFactor( 0x000d,      0.0002012 ), /* (15 3) */
},
#endif

#ifdef SubDiv16
{
    scBezFactor( 0xd2f0,      0.8239746 ), /* (16 0) */
    scBezFactor( 0x2a30,      0.1647949 ), /* (16 1) */
    scBezFactor( 0x02d0,      0.0109863 ), /* (16 2) */
    scBezFactor( 0x0010,      0.0002441 ), /* (16 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xd04f,      0.8137178 ), /* (17 0) */
    scBezFactor( 0x2c73,      0.1736385 ), /* (17 1) */
    scBezFactor( 0x0329,      0.0123509 ), /* (17 2) */
    scBezFactor( 0x0013,      0.0002928 ), /* (17 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0xcdb5,      0.8035464 ), /* (18 0) */
    scBezFactor( 0x2eac,      0.1823173 ), /* (18 1) */
    scBezFactor( 0x0387,      0.0137887 ), /* (18 2) */
    scBezFactor( 0x0016,      0.0003476 ), /* (18 3) */
},
#endif

#ifdef SubDiv256
```



```
{
    scBezFactor( 0xcb20,      0.7954602 ), /* (19 0) */
    scBezFactor( 0x30da,      0.1908322 ), /* (19 1) */
    scBezFactor( 0x03ea,      0.0152988 ), /* (19 2) */
    scBezFactor( 0x001a,      0.0004088 ) /* (19 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0xc890,      0.7834587 ), /* (20 0) */
    scBezFactor( 0x32fd,      0.1991844 ), /* (20 1) */
    scBezFactor( 0x0452,      0.0168800 ), /* (20 2) */
    scBezFactor( 0x001f,      0.0004768 ) /* (20 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xc606,      0.7735416 ), /* (21 0) */
    scBezFactor( 0x3516,      0.2073750 ), /* (21 1) */
    scBezFactor( 0x04be,      0.0185314 ), /* (21 2) */
    scBezFactor( 0x0024,      0.0005520 ) /* (21 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0xc382,      0.7637086 ), /* (22 0) */
    scBezFactor( 0x3724,      0.2154050 ), /* (22 1) */
    scBezFactor( 0x052f,      0.0202518 ), /* (22 2) */
    scBezFactor( 0x0029,      0.0006347 ) /* (22 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xc103,      0.7539592 ), /* (23 0) */
    scBezFactor( 0x3928,      0.2232755 ), /* (23 1) */
    scBezFactor( 0x05a4,      0.0220401 ), /* (23 2) */
    scBezFactor( 0x002f,      0.0007252 ) /* (23 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0xbe8a,      0.7442932 ), /* (24 0) */
    scBezFactor( 0x3b22,      0.2309875 ), /* (24 1) */
    scBezFactor( 0x061e,      0.0238953 ), /* (24 2) */
    scBezFactor( 0x0036,      0.0008240 ) /* (24 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xbc15,      0.7347102 ), /* (25 0) */
    scBezFactor( 0x3d11,      0.2385423 ), /* (25 1) */
    scBezFactor( 0x069b,      0.0258163 ), /* (25 2) */
    scBezFactor( 0x003d,      0.0009313 ) /* (25 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0xb9a7,      0.7252097 ), /* (26 0) */
    scBezFactor( 0x3ef5,      0.2459407 ), /* (26 1) */
}
```

```
    scBezFactor( 0x071e,      0.9020 ), /* (26 2) */
    scBezFactor( 0x0044,      0.0010476 ), /* (26 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xb73e,      0.7157915 ), /* (27 0) */
    scBezFactor( 0x40d0,      0.2531839 ), /* (27 1) */
    scBezFactor( 0x07a4,      0.0298514 ), /* (27 2) */
    scBezFactor( 0x004c,      0.0011732 ), /* (27 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0xb4da,      0.7064552 ), /* (28 0) */
    scBezFactor( 0x42a1,      0.2602730 ), /* (28 1) */
    scBezFactor( 0x082e,      0.0319633 ), /* (28 2) */
    scBezFactor( 0x0055,      0.0013084 ), /* (28 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xb27b,      0.6972005 ), /* (29 0) */
    scBezFactor( 0x4467,      0.2672090 ), /* (29 1) */
    scBezFactor( 0x08bd,      0.0341368 ), /* (29 2) */
    scBezFactor( 0x005f,      0.0014537 ), /* (29 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0xb022,      0.6880269 ), /* (30 0) */
    scBezFactor( 0x4624,      0.2739930 ), /* (30 1) */
    scBezFactor( 0x094f,      0.0363708 ), /* (30 2) */
    scBezFactor( 0x0069,      0.0016093 ), /* (30 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xadce,      0.6789342 ), /* (31 0) */
    scBezFactor( 0x47d7,      0.2806261 ), /* (31 1) */
    scBezFactor( 0x09e5,      0.0386640 ), /* (31 2) */
    scBezFactor( 0x0074,      0.0017757 ), /* (31 3) */
},
#endif

#ifdef SubDiv8
{
    scBezFactor( 0xab80,      0.6699219 ), /* (32 0) */
    scBezFactor( 0x4980,      0.2871094 ), /* (32 1) */
    scBezFactor( 0x0a80,      0.0410156 ), /* (32 2) */
    scBezFactor( 0x0080,      0.0019531 ), /* (32 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0xa936,      0.6609897 ), /* (33 0) */
    scBezFactor( 0x4b1f,      0.2934439 ), /* (33 1) */
    scBezFactor( 0x0b1d,      0.0434244 ), /* (33 2) */
    scBezFactor( 0x008c,      0.0021420 ), /* (33 3) */
},
}
```

#endif

#ifdef SubDiv128

```
{
    scBezFactor( 0xa6f2,    0.6521373 ), /* (34 0) */
    scBezFactor( 0x4cb4,    0.2996306 ), /* (34 1) */
    scBezFactor( 0x0bbf,    0.0458894 ), /* (34 2) */
    scBezFactor( 0x0099,    0.0023427 ) /* (34 3) */
},
#endif
```

#ifdef SubDiv256

```
{
    scBezFactor( 0xa4b3,    0.6433643 ), /* (35 0) */
    scBezFactor( 0x4e40,    0.3056708 ), /* (35 1) */
    scBezFactor( 0x0c64,    0.0484094 ), /* (35 2) */
    scBezFactor( 0x00a7,    0.0025555 ) /* (35 3) */
},
#endif
```

#ifdef SubDiv64

```
{
    scBezFactor( 0xa279,    0.6346703 ), /* (36 0) */
    scBezFactor( 0x4fc2,    0.3115654 ), /* (36 1) */
    scBezFactor( 0x0d0d,    0.0509834 ), /* (36 2) */
    scBezFactor( 0x00b6,    0.0027809 ) /* (36 3) */
},
#endif
```

#ifdef SubDiv256

```
{
    scBezFactor( 0xa045,    0.6260549 ), /* (37 0) */
    scBezFactor( 0x513b,    0.3173155 ), /* (37 1) */
    scBezFactor( 0x0db9,    0.0536104 ), /* (37 2) */
    scBezFactor( 0x00c5,    0.0030192 ) /* (37 3) */
},
#endif
```

#ifdef SubDiv128

```
{
    scBezFactor( 0x9e15,    0.6175179 ), /* (38 0) */
    scBezFactor( 0x52ab,    0.3229222 ), /* (38 1) */
    scBezFactor( 0x0e68,    0.0562892 ), /* (38 2) */
    scBezFactor( 0x00d6,    0.0032706 ) /* (38 3) */
},
#endif
```

#ifdef SubDiv256

```
{
    scBezFactor( 0x9beb,    0.6090589 ), /* (39 0) */
    scBezFactor( 0x5411,    0.3283866 ), /* (39 1) */
    scBezFactor( 0x0f1b,    0.0590188 ), /* (39 2) */
    scBezFactor( 0x00e7,    0.0035357 ) /* (39 3) */
},
#endif
```

#ifdef SubDiv32

```
{
    scBezFactor( 0x99c6,    0.6006775 ), /* (40 0) */
    scBezFactor( 0x556e,    0.3337097 ), /* (40 1) */
    scBezFactor( 0x0fd2,    0.0617981 ), /* (40 2) */
    scBezFactor( 0x00fa,    0.0038147 ) /* (40 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x97a5,      0.5923733 ), /* (41 0) */
    scBezFactor( 0x56c1,      0.3388926 ), /* (41 1) */
    scBezFactor( 0x108b,      0.0646260 ), /* (41 2) */
    scBezFactor( 0x010d,      0.0041080 ), /* (41 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x958a,      0.5841460 ), /* (42 0) */
    scBezFactor( 0x580c,      0.3439364 ), /* (42 1) */
    scBezFactor( 0x1147,      0.0675015 ), /* (42 2) */
    scBezFactor( 0x0121,      0.0044160 ), /* (42 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x9374,      0.5759953 ), /* (43 0) */
    scBezFactor( 0x594d,      0.3488422 ), /* (43 1) */
    scBezFactor( 0x1207,      0.0704235 ), /* (43 2) */
    scBezFactor( 0x0136,      0.0047390 ), /* (43 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x9163,      0.5679207 ), /* (44 0) */
    scBezFactor( 0x5a86,      0.3536110 ), /* (44 1) */
    scBezFactor( 0x12c9,      0.0733910 ), /* (44 2) */
    scBezFactor( 0x014c,      0.0050774 ), /* (44 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x8f57,      0.5599219 ), /* (45 0) */
    scBezFactor( 0x5bb5,      0.3582439 ), /* (45 1) */
    scBezFactor( 0x138f,      0.0764027 ), /* (45 2) */
    scBezFactor( 0x0163,      0.0054315 ), /* (45 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x8d4f,      0.5519986 ), /* (46 0) */
    scBezFactor( 0x5cdc,      0.3627419 ), /* (46 1) */
    scBezFactor( 0x1457,      0.0794578 ), /* (46 2) */
    scBezFactor( 0x017c,      0.0058017 ), /* (46 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x8b4d,      0.5441504 ), /* (47 0) */
    scBezFactor( 0x5dfa,      0.3671063 ), /* (47 1) */
    scBezFactor( 0x1522,      0.0825550 ), /* (47 2) */
    scBezFactor( 0x0195,      0.0061883 ), /* (47 3) */
},
#endif

#ifdef SubDiv16
{
    scBezFactor( 0x8950,      0.5363770 ), /* (48 0) */

```

```
    scBezFactor( 0x5f10,      0.000379 ), /* (48 1) */
    scBezFactor( 0x15f0,      0.0856934 ), /* (48 2) */
    scBezFactor( 0x01b0,      0.0065918 ), /* (48 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x8757,      0.5286779 ), /* (49 0) */
    scBezFactor( 0x601c,      0.3754379 ), /* (49 1) */
    scBezFactor( 0x16c0,      0.0888718 ), /* (49 2) */
    scBezFactor( 0x01cb,      0.0070124 ), /* (49 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x8563,      0.5210528 ), /* (50 0) */
    scBezFactor( 0x6120,      0.3794074 ), /* (50 1) */
    scBezFactor( 0x1793,      0.0920892 ), /* (50 2) */
    scBezFactor( 0x01e8,      0.0074506 ), /* (50 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x8374,      0.5135015 ), /* (51 0) */
    scBezFactor( 0x6120,      0.3832474 ), /* (51 1) */
    scBezFactor( 0x1868,      0.0953445 ), /* (51 2) */
    scBezFactor( 0x0206,      0.0079066 ), /* (51 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x818a,      0.5060234 ), /* (52 0) */
    scBezFactor( 0x630f,      0.3869591 ), /* (52 1) */
    scBezFactor( 0x1940,      0.0986366 ), /* (52 2) */
    scBezFactor( 0x0225,      0.0083809 ), /* (52 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x7fa5,      0.4986183 ), /* (53 0) */
    scBezFactor( 0x63fa,      0.3905434 ), /* (53 1) */
    scBezFactor( 0x1a1a,      0.1019645 ), /* (53 2) */
    scBezFactor( 0x0245,      0.0088738 ), /* (53 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x7dc4,      0.4912858 ), /* (54 0) */
    scBezFactor( 0x64dd,      0.3940015 ), /* (54 1) */
    scBezFactor( 0x1af6,      0.1053271 ), /* (54 2) */
    scBezFactor( 0x0267,      0.0093856 ), /* (54 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x7be9,      0.4840255 ), /* (55 0) */
    scBezFactor( 0x65b7,      0.3973344 ), /* (55 1) */
    scBezFactor( 0x1bd5,      0.1087233 ), /* (55 2) */
    scBezFactor( 0x0289,      0.0099167 ), /* (55 3) */
}
```

```
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0x7a12,    0.4768372 ), /* (56 0) */
    scBezFactor( 0x668a,    0.4005432 ), /* (56 1) */
    scBezFactor( 0x1cb6,    0.1121521 ), /* (56 2) */
    scBezFactor( 0x02ae,    0.0104675 ), /* (56 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x783f,    0.4697203 ), /* (57 0) */
    scBezFactor( 0x6754,    0.4036290 ), /* (57 1) */
    scBezFactor( 0x1d98,    0.1156123 ), /* (57 2) */
    scBezFactor( 0x02d3,    0.0110384 ), /* (57 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x7671,    0.4626746 ), /* (58 0) */
    scBezFactor( 0x6816,    0.4065928 ), /* (58 1) */
    scBezFactor( 0x1e7d,    0.1191030 ), /* (58 2) */
    scBezFactor( 0x02fa,    0.0116296 ), /* (58 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x74a8,    0.4556997 ), /* (59 0) */
    scBezFactor( 0x68d0,    0.4094358 ), /* (59 1) */
    scBezFactor( 0x1f64,    0.1226229 ), /* (59 2) */
    scBezFactor( 0x0322,    0.0122415 ), /* (59 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x72e4,    0.4487953 ), /* (60 0) */
    scBezFactor( 0x6983,    0.4121590 ), /* (60 1) */
    scBezFactor( 0x204c,    0.1261711 ), /* (60 2) */
    scBezFactor( 0x034b,    0.0128746 ), /* (60 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x7124,    0.4419610 ), /* (61 0) */
    scBezFactor( 0x6a2d,    0.4147634 ), /* (61 1) */
    scBezFactor( 0x2137,    0.1297465 ), /* (61 2) */
    scBezFactor( 0x0376,    0.0135291 ), /* (61 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x6f69,    0.4351964 ), /* (62 0) */
    scBezFactor( 0x6ad0,    0.4172502 ), /* (62 1) */
    scBezFactor( 0x2223,    0.1333480 ), /* (62 2) */
    scBezFactor( 0x03a2,    0.0142055 ), /* (62 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x6db2,      0.4285012 ), /* (63 0) */
    scBezFactor( 0x6b6c,      0.4196203 ), /* (63 1) */
    scBezFactor( 0x2310,      0.1369745 ), /* (63 2) */
    scBezFactor( 0x03d0,      0.0149040 ), /* (63 3) */
},
#endif
```

```
#ifdef SubDiv4
{
    scBezFactor( 0x6c00,      0.4218750 ), /* (64 0) */
    scBezFactor( 0x6c00,      0.4218750 ), /* (64 1) */
    scBezFactor( 0x2400,      0.1406250 ), /* (64 2) */
    scBezFactor( 0x0400,      0.0156250 ), /* (64 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x6a52,      0.4153175 ), /* (65 0) */
    scBezFactor( 0x6c8c,      0.4240152 ), /* (65 1) */
    scBezFactor( 0x24f0,      0.1442984 ), /* (65 2) */
    scBezFactor( 0x0430,      0.0163689 ), /* (65 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x68a8,      0.4088283 ), /* (66 0) */
    scBezFactor( 0x6d11,      0.4260421 ), /* (66 1) */
    scBezFactor( 0x25e2,      0.1479936 ), /* (66 2) */
    scBezFactor( 0x0463,      0.0171361 ), /* (66 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x6704,      0.4024070 ), /* (67 0) */
    scBezFactor( 0x6d8e,      0.4279566 ), /* (67 1) */
    scBezFactor( 0x26d6,      0.1517095 ), /* (67 2) */
    scBezFactor( 0x0496,      0.0179269 ), /* (67 3) */
},
#endif
```

```
#ifdef SubDiv64
{
    scBezFactor( 0x6563,      0.3960533 ), /* (68 0) */
    scBezFactor( 0x6e04,      0.4297600 ), /* (68 1) */
    scBezFactor( 0x27cb,      0.1554451 ), /* (68 2) */
    scBezFactor( 0x04cc,      0.0187416 ), /* (68 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x63c7,      0.3897669 ), /* (69 0) */
    scBezFactor( 0x6e73,      0.4314532 ), /* (69 1) */
    scBezFactor( 0x28c1,      0.1591993 ), /* (69 2) */
    scBezFactor( 0x0503,      0.0195807 ), /* (69 3) */
},
#endif
```

```
#ifdef SubDiv128
{
```

```
    scBezFactor( 0x6230,    0.4473 ), /* (70 0) */
    scBezFactor( 0x6edb,    0.4330373 ), /* (70 1) */
    scBezFactor( 0x29b8,    0.1629710 ), /* (70 2) */
    scBezFactor( 0x053b,    0.0204444 ) /* (70 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x609c,    0.3773943 ), /* (71 0) */
    scBezFactor( 0x6f3c,    0.4345134 ), /* (71 1) */
    scBezFactor( 0x2ab0,    0.1667592 ), /* (71 2) */
    scBezFactor( 0x0576,    0.0213332 ) /* (71 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0x5f0e,    0.3713074 ), /* (72 0) */
    scBezFactor( 0x6f96,    0.4358826 ), /* (72 1) */
    scBezFactor( 0x2baa,    0.1705627 ), /* (72 2) */
    scBezFactor( 0x05b2,    0.0222473 ) /* (72 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x5d83,    0.3652863 ), /* (73 0) */
    scBezFactor( 0x6fe8,    0.4371459 ), /* (73 1) */
    scBezFactor( 0x2ca4,    0.1743806 ), /* (73 2) */
    scBezFactor( 0x05ef,    0.0231872 ) /* (73 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x5bfd,    0.3593307 ), /* (74 0) */
    scBezFactor( 0x7034,    0.4383044 ), /* (74 1) */
    scBezFactor( 0x2d9f,    0.1782117 ), /* (74 2) */
    scBezFactor( 0x062e,    0.0241532 ) /* (74 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x5a7b,    0.3534401 ), /* (75 0) */
    scBezFactor( 0x7079,    0.4393592 ), /* (75 1) */
    scBezFactor( 0x2e9b,    0.1820549 ), /* (75 2) */
    scBezFactor( 0x066f,    0.0251457 ) /* (75 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x58fd,    0.3476143 ), /* (76 0) */
    scBezFactor( 0x70b8,    0.4403114 ), /* (76 1) */
    scBezFactor( 0x2f97,    0.1859093 ), /* (76 2) */
    scBezFactor( 0x06b2,    0.0261650 ) /* (76 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x5783,    0.3418528 ), /* (77 0) */
    scBezFactor( 0x70ef,    0.4411620 ), /* (77 1) */
    scBezFactor( 0x3095,    0.1897736 ), /* (77 2) */
}
```



```
    scBezFactor( 0x06f7,      0.115 ), /* (77 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x560e,      0.3361554 ), /* (78 0) */
    scBezFactor( 0x7121,      0.4419122 ), /* (78 1) */
    scBezFactor( 0x3192,      0.1936469 ), /* (78 2) */
    scBezFactor( 0x073d,      0.0282855 ), /* (78 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x549d,      0.3305216 ), /* (79 0) */
    scBezFactor( 0x714b,      0.4425629 ), /* (79 1) */
    scBezFactor( 0x3291,      0.1975281 ), /* (79 2) */
    scBezFactor( 0x0785,      0.0293874 ), /* (79 3) */
},
#endif

#ifdef SubDiv16
{
    scBezFactor( 0x5330,      0.3249512 ), /* (80 0) */
    scBezFactor( 0x7170,      0.4431152 ), /* (80 1) */
    scBezFactor( 0x3390,      0.2014160 ), /* (80 2) */
    scBezFactor( 0x07d0,      0.0305176 ), /* (80 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x51c7,      0.3194436 ), /* (81 0) */
    scBezFactor( 0x718d,      0.4435703 ), /* (81 1) */
    scBezFactor( 0x348f,      0.2053097 ), /* (81 2) */
    scBezFactor( 0x081b,      0.0316764 ), /* (81 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x5062,      0.3139987 ), /* (82 0) */
    scBezFactor( 0x71a5,      0.4439292 ), /* (82 1) */
    scBezFactor( 0x358e,      0.2092080 ), /* (82 2) */
    scBezFactor( 0x0869,      0.0328641 ), /* (82 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x4f01,      0.3086160 ), /* (83 0) */
    scBezFactor( 0x71b6,      0.4441929 ), /* (83 1) */
    scBezFactor( 0x368e,      0.2131099 ), /* (83 2) */
    scBezFactor( 0x08b9,      0.0340812 ), /* (83 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x4da4,      0.3032951 ), /* (84 0) */
    scBezFactor( 0x71c1,      0.4443626 ), /* (84 1) */
    scBezFactor( 0x378e,      0.2170143 ), /* (84 2) */
    scBezFactor( 0x090b,      0.0353279 ), /* (84 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x4c4c,      0.2980358 ), /* (85 0) */
    scBezFactor( 0x71c6,      0.4444394 ), /* (85 1) */
    scBezFactor( 0x388e,      0.2209201 ), /* (85 2) */
    scBezFactor( 0x095e,      0.0366047 ), /* (85 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x4af7,      0.2928376 ), /* (86 0) */
    scBezFactor( 0x71c5,      0.4444242 ), /* (86 1) */
    scBezFactor( 0x398e,      0.2248263 ), /* (86 2) */
    scBezFactor( 0x09b4,      0.0379119 ), /* (86 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x49a6,      0.2877002 ), /* (87 0) */
    scBezFactor( 0x71be,      0.4443181 ), /* (87 1) */
    scBezFactor( 0x3a8e,      0.2287318 ), /* (87 2) */
    scBezFactor( 0x0a0c,      0.0392498 ), /* (87 3) */
},
#endif
```

```
#ifdef SubDiv32
{
    scBezFactor( 0x485a,      0.2826233 ), /* (88 0) */
    scBezFactor( 0x71b2,      0.4441223 ), /* (88 1) */
    scBezFactor( 0x3b8e,      0.2326355 ), /* (88 2) */
    scBezFactor( 0x0a66,      0.0406189 ), /* (88 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x4711,      0.2776064 ), /* (89 0) */
    scBezFactor( 0x719f,      0.4438378 ), /* (89 1) */
    scBezFactor( 0x3c8d,      0.2365363 ), /* (89 2) */
    scBezFactor( 0x0ac1,      0.0420194 ), /* (89 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x45cc,      0.2726493 ), /* (90 0) */
    scBezFactor( 0x7186,      0.4434657 ), /* (90 1) */
    scBezFactor( 0x3d8d,      0.2404332 ), /* (90 2) */
    scBezFactor( 0x0b1f,      0.0434518 ), /* (90 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x448b,      0.2677515 ), /* (91 0) */
    scBezFactor( 0x7168,      0.4430071 ), /* (91 1) */
    scBezFactor( 0x3e8c,      0.2443251 ), /* (91 2) */
    scBezFactor( 0x0b7f,      0.0449163 ), /* (91 3) */
},
#endif
```

```
#ifdef SubDiv64
```

```
{
    scBezFactor( 0x434e,      0.2829128 ), /* (92 0) */
    scBezFactor( 0x7145,      0.4424629 ), /* (92 1) */
    scBezFactor( 0x3f8a,      0.2482109 ), /* (92 2) */
    scBezFactor( 0x0be1,      0.0464134 ), /* (92 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x4214,      0.2581326 ), /* (93 0) */
    scBezFactor( 0x711c,      0.4418344 ), /* (93 1) */
    scBezFactor( 0x4088,      0.2520896 ), /* (93 2) */
    scBezFactor( 0x0c46,      0.0479434 ), /* (93 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x40df,      0.2534108 ), /* (94 0) */
    scBezFactor( 0x70ed,      0.4411225 ), /* (94 1) */
    scBezFactor( 0x4186,      0.2559600 ), /* (94 2) */
    scBezFactor( 0x0cac,      0.0495067 ), /* (94 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x3fad,      0.2487469 ), /* (95 0) */
    scBezFactor( 0x70b9,      0.4403284 ), /* (95 1) */
    scBezFactor( 0x4283,      0.2598211 ), /* (95 2) */
    scBezFactor( 0x0d15,      0.0511035 ), /* (95 3) */
},
#endif

#ifdef SubDiv8
{
    scBezFactor( 0x3e80,      0.2441406 ), /* (96 0) */
    scBezFactor( 0x7080,      0.4394531 ), /* (96 1) */
    scBezFactor( 0x4380,      0.2636719 ), /* (96 2) */
    scBezFactor( 0x0d80,      0.0527344 ), /* (96 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x3d55,      0.2395915 ), /* (97 0) */
    scBezFactor( 0x7041,      0.4384977 ), /* (97 1) */
    scBezFactor( 0x447b,      0.2675112 ), /* (97 2) */
    scBezFactor( 0x0ded,      0.0543995 ), /* (97 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x3c2f,      0.2350993 ), /* (98 0) */
    scBezFactor( 0x6ffd,      0.4374633 ), /* (98 1) */
    scBezFactor( 0x4576,      0.2713380 ), /* (98 2) */
    scBezFactor( 0x0e5c,      0.0560994 ), /* (98 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x3b0c,      0.2306636 ), /* (99 0) */
    scBezFactor( 0x6fb4,      0.4363509 ), /* (99 1) */
}
```

```
    scBezFactor( 0x4670,      0.512 ), /* (99 2) */
    scBezFactor( 0x0e0e,      0.0578343 ), /* (99 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x39ed,      0.2262840 ), /* (100 0) */
    scBezFactor( 0x6f66,      0.4351616 ), /* (100 1) */
    scBezFactor( 0x4769,      0.2789497 ), /* (100 2) */
    scBezFactor( 0x0f42,      0.0596046 ), /* (100 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x38d2,      0.2219602 ), /* (101 0) */
    scBezFactor( 0x6f13,      0.4338965 ), /* (101 1) */
    scBezFactor( 0x4861,      0.2827325 ), /* (101 2) */
    scBezFactor( 0x0fb8,      0.0614107 ), /* (101 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x37ba,      0.2176919 ), /* (102 0) */
    scBezFactor( 0x6e0c,      0.4325566 ), /* (102 1) */
    scBezFactor( 0x4957,      0.2864985 ), /* (102 2) */
    scBezFactor( 0x1031,      0.0632529 ), /* (102 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x36a6,      0.2134786 ), /* (103 0) */
    scBezFactor( 0x6e5f,      0.4311431 ), /* (103 1) */
    scBezFactor( 0x4a4d,      0.2902467 ), /* (103 2) */
    scBezFactor( 0x10ac,      0.0651316 ), /* (103 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0x3596,      0.2093201 ), /* (104 0) */
    scBezFactor( 0x6dfe,      0.4296570 ), /* (104 1) */
    scBezFactor( 0x4b42,      0.2939758 ), /* (104 2) */
    scBezFactor( 0x112a,      0.0670471 ), /* (104 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x3489,      0.2052159 ), /* (105 0) */
    scBezFactor( 0x6d97,      0.4280993 ), /* (105 1) */
    scBezFactor( 0x4c35,      0.2976850 ), /* (105 2) */
    scBezFactor( 0x11a9,      0.0689998 ), /* (105 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x337f,      0.2011657 ), /* (106 0) */
    scBezFactor( 0x6d2d,      0.4264712 ), /* (106 1) */
    scBezFactor( 0x4d26,      0.3013730 ), /* (106 2) */
    scBezFactor( 0x122c,      0.0709901 ), /* (106 3) */
},
#endif
```

#endif

#ifdef SubDiv256

```
{
    scBezFactor( 0x3279,    0.1971691 ), /* (107 0) */
    scBezFactor( 0x6cbd,    0.4247738 ), /* (107 1) */
    scBezFactor( 0x4e17,    0.3050389 ), /* (107 2) */
    scBezFactor( 0x12b1,    0.0730183 ) /* (107 3) */
},
```

#endif

#ifdef SubDiv64

```
{
    scBezFactor( 0x3177,    0.1932259 ), /* (108 0) */
    scBezFactor( 0x6c4a,    0.4230080 ), /* (108 1) */
    scBezFactor( 0x4f05,    0.3086815 ), /* (108 2) */
    scBezFactor( 0x1338,    0.0750847 ) /* (108 3) */
},
```

#endif

#ifdef SubDiv256

```
{
    scBezFactor( 0x3078,    0.1893355 ), /* (109 0) */
    scBezFactor( 0x6bd2,    0.4211749 ), /* (109 1) */
    scBezFactor( 0x4ff2,    0.3122998 ), /* (109 2) */
    scBezFactor( 0x13c2,    0.0771897 ) /* (109 3) */
},
```

#endif

#ifdef SubDiv128

```
{
    scBezFactor( 0x2f7c,    0.1854978 ), /* (110 0) */
    scBezFactor( 0x6b55,    0.4192758 ), /* (110 1) */
    scBezFactor( 0x50de,    0.3158927 ), /* (110 2) */
    scBezFactor( 0x144f,    0.0793338 ) /* (110 3) */
},
```

#endif

#ifdef SubDiv256

```
{
    scBezFactor( 0x2e84,    0.1817122 ), /* (111 0) */
    scBezFactor( 0x6ad4,    0.4173115 ), /* (111 1) */
    scBezFactor( 0x51c8,    0.3194591 ), /* (111 2) */
    scBezFactor( 0x14de,    0.0815172 ) /* (111 3) */
},
```

#endif

#ifdef SubDiv16

```
{
    scBezFactor( 0x2d90,    0.1779785 ), /* (112 0) */
    scBezFactor( 0x6a50,    0.4152832 ), /* (112 1) */
    scBezFactor( 0x52b0,    0.3229980 ), /* (112 2) */
    scBezFactor( 0x1570,    0.0837402 ) /* (112 3) */
},
```

#endif

#ifdef SubDiv256

```
{
    scBezFactor( 0x2c9e,    0.1742963 ), /* (113 0) */
    scBezFactor( 0x69c6,    0.4131920 ), /* (113 1) */
    scBezFactor( 0x5396,    0.3265083 ), /* (113 2) */
    scBezFactor( 0x1604,    0.0860034 ) /* (113 3) */
},
```

#endif

```

#ifdef SubDiv128
{
    scBezFactor( 0x2bb0,    0.1706653 ),    /* (114 0) */
    scBezFactor( 0x6939,    0.4110389 ),    /* (114 1) */
    scBezFactor( 0x547a,    0.3299890 ),    /* (114 2) */
    scBezFactor( 0x169b,    0.0883069 )     /* (114 3) */
},
#endif

```

```

#ifdef SubDiv256
{
    scBezFactor( 0x2ac6,    0.1670850 ),    /* (115 0) */
    scBezFactor( 0x68a8,    0.4088250 ),    /* (115 1) */
    scBezFactor( 0x555c,    0.3334388 ),    /* (115 2) */
    scBezFactor( 0x1734,    0.0906512 )     /* (115 3) */
},
#endif

```

```

#ifdef SubDiv64
{
    scBezFactor( 0x29de,    0.1635551 ),    /* (116 0) */
    scBezFactor( 0x6813,    0.4065514 ),    /* (116 1) */
    scBezFactor( 0x563c,    0.3368568 ),    /* (116 2) */
    scBezFactor( 0x17d1,    0.0930367 )     /* (116 3) */
},
#endif

```

```

#ifdef SubDiv256
{
    scBezFactor( 0x28fa,    0.1600754 ),    /* (117 0) */
    scBezFactor( 0x677a,    0.4042191 ),    /* (117 1) */
    scBezFactor( 0x571a,    0.3402420 ),    /* (117 2) */
    scBezFactor( 0x1870,    0.0954636 )     /* (117 3) */
},
#endif

```

```

#ifdef SubDiv128
{
    scBezFactor( 0x2819,    0.1566453 ),    /* (118 0) */
    scBezFactor( 0x66de,    0.4018292 ),    /* (118 1) */
    scBezFactor( 0x57f5,    0.3435931 ),    /* (118 2) */
    scBezFactor( 0x1912,    0.0979323 )     /* (118 3) */
},
#endif

```

```

#ifdef SubDiv256
{
    scBezFactor( 0x273c,    0.1532646 ),    /* (119 0) */
    scBezFactor( 0x663d,    0.3993829 ),    /* (119 1) */
    scBezFactor( 0x58cf,    0.3469092 ),    /* (119 2) */
    scBezFactor( 0x19b6,    0.1004433 )     /* (119 3) */
},
#endif

```

```

#ifdef SubDiv32
{
    scBezFactor( 0x2662,    0.1499329 ),    /* (120 0) */
    scBezFactor( 0x659a,    0.3968811 ),    /* (120 1) */
    scBezFactor( 0x59a6,    0.3501892 ),    /* (120 2) */
    scBezFactor( 0x1a5e,    0.1029968 )     /* (120 3) */
},
#endif

```

```

#ifdef SubDiv256
{
    scBezFactor( 0x258a,    0.1466498 ),    /* (121 0) */

```

```
    scBezFactor( 0x64f2,      0.1250000 ), /* (121 1) */
    scBezFactor( 0x5a7a,      0.3534320 ), /* (121 2) */
    scBezFactor( 0x1b08,      0.1055933 ), /* (121 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x24b6,      0.1434150 ), /* (122 0) */
    scBezFactor( 0x6447,      0.3917155 ), /* (122 1) */
    scBezFactor( 0x5b4c,      0.3566365 ), /* (122 2) */
    scBezFactor( 0x1bb5,      0.1082330 ), /* (122 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x23e5,      0.1402281 ), /* (123 0) */
    scBezFactor( 0x6399,      0.3890539 ), /* (123 1) */
    scBezFactor( 0x5c1b,      0.3598017 ), /* (123 2) */
    scBezFactor( 0x1c65,      0.1109163 ), /* (123 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x2318,      0.1370888 ), /* (124 0) */
    scBezFactor( 0x62e7,      0.3863411 ), /* (124 1) */
    scBezFactor( 0x5ce8,      0.3629265 ), /* (124 2) */
    scBezFactor( 0x1d17,      0.1136436 ), /* (124 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x224d,      0.1339967 ), /* (125 0) */
    scBezFactor( 0x6232,      0.3835782 ), /* (125 1) */
    scBezFactor( 0x5db2,      0.3660098 ), /* (125 2) */
    scBezFactor( 0x1dcd,      0.1164153 ), /* (125 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x2186,      0.1309514 ), /* (126 0) */
    scBezFactor( 0x6179,      0.3807664 ), /* (126 1) */
    scBezFactor( 0x5e7a,      0.3690505 ), /* (126 2) */
    scBezFactor( 0x1e85,      0.1192317 ), /* (126 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x20c1,      0.1279526 ), /* (127 0) */
    scBezFactor( 0x60be,      0.3779066 ), /* (127 1) */
    scBezFactor( 0x5f3e,      0.3720476 ), /* (127 2) */
    scBezFactor( 0x1f41,      0.1220931 ), /* (127 3) */
},
#endif

#ifdef SubDiv2
{
    scBezFactor( 0x2000,      0.1250000 ), /* (128 0) */
    scBezFactor( 0x6000,      0.3750000 ), /* (128 1) */
    scBezFactor( 0x6000,      0.3750000 ), /* (128 2) */
    scBezFactor( 0x2000,      0.1250000 ), /* (128 3) */
}
```

```
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x1f41,      0.1220931 ), /* (129 0) */
    scBezFactor( 0x5f3e,      0.3720476 ), /* (129 1) */
    scBezFactor( 0x60be,      0.3779066 ), /* (129 2) */
    scBezFactor( 0x20c1,      0.1279526 ), /* (129 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x1e85,      0.1192317 ), /* (130 0) */
    scBezFactor( 0x5e7a,      0.3690505 ), /* (130 1) */
    scBezFactor( 0x6179,      0.3807664 ), /* (130 2) */
    scBezFactor( 0x2186,      0.1309514 ), /* (130 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x1dcd,      0.1164153 ), /* (131 0) */
    scBezFactor( 0x5db2,      0.3660098 ), /* (131 1) */
    scBezFactor( 0x6232,      0.3835782 ), /* (131 2) */
    scBezFactor( 0x224d,      0.1339967 ), /* (131 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x1d17,      0.1136436 ), /* (132 0) */
    scBezFactor( 0x5ce8,      0.3629265 ), /* (132 1) */
    scBezFactor( 0x62e7,      0.3863411 ), /* (132 2) */
    scBezFactor( 0x2318,      0.1370888 ), /* (132 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x1c65,      0.1109163 ), /* (133 0) */
    scBezFactor( 0x5c1b,      0.3598017 ), /* (133 1) */
    scBezFactor( 0x6399,      0.3890539 ), /* (133 2) */
    scBezFactor( 0x23e5,      0.1402281 ), /* (133 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x1bb5,      0.1082330 ), /* (134 0) */
    scBezFactor( 0x5b4c,      0.3566365 ), /* (134 1) */
    scBezFactor( 0x6447,      0.3917155 ), /* (134 2) */
    scBezFactor( 0x24b6,      0.1434150 ), /* (134 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x1b08,      0.1055933 ), /* (135 0) */
    scBezFactor( 0x5a7a,      0.3534320 ), /* (135 1) */
    scBezFactor( 0x64f2,      0.3943250 ), /* (135 2) */
    scBezFactor( 0x258a,      0.1466498 ), /* (135 3) */
},
#endif
```



```
#ifdef SubDiv32
{
    scBezFactor( 0x1a5e, 0.1029968 ), /* (136 0) */
    scBezFactor( 0x59a6, 0.3501892 ), /* (136 1) */
    scBezFactor( 0x659a, 0.3968811 ), /* (136 2) */
    scBezFactor( 0x2662, 0.1499329 ), /* (136 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x19b6, 0.1004433 ), /* (137 0) */
    scBezFactor( 0x58cf, 0.3469092 ), /* (137 1) */
    scBezFactor( 0x663d, 0.3993829 ), /* (137 2) */
    scBezFactor( 0x273c, 0.1532646 ), /* (137 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x1912, 0.0979323 ), /* (138 0) */
    scBezFactor( 0x57f5, 0.3435931 ), /* (138 1) */
    scBezFactor( 0x66de, 0.4018292 ), /* (138 2) */
    scBezFactor( 0x2819, 0.1566453 ), /* (138 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x1870, 0.0954636 ), /* (139 0) */
    scBezFactor( 0x571a, 0.3402420 ), /* (139 1) */
    scBezFactor( 0x677a, 0.4042191 ), /* (139 2) */
    scBezFactor( 0x28fa, 0.1600754 ), /* (139 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x17d1, 0.0930367 ), /* (140 0) */
    scBezFactor( 0x563c, 0.3368568 ), /* (140 1) */
    scBezFactor( 0x6813, 0.4065514 ), /* (140 2) */
    scBezFactor( 0x29de, 0.1635551 ), /* (140 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x1734, 0.0906512 ), /* (141 0) */
    scBezFactor( 0x555c, 0.3334388 ), /* (141 1) */
    scBezFactor( 0x68a8, 0.4088250 ), /* (141 2) */
    scBezFactor( 0x2ac6, 0.1670850 ), /* (141 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x169b, 0.0883069 ), /* (142 0) */
    scBezFactor( 0x547a, 0.3299890 ), /* (142 1) */
    scBezFactor( 0x6939, 0.4110389 ), /* (142 2) */
    scBezFactor( 0x2bb0, 0.1706653 ), /* (142 3) */
},
#endif

#ifdef SubDiv256
{
```

```
    scBezFactor( 0x1604,    0.00000034 ), /* (143 0) */
    scBezFactor( 0x5396,    0.3265083 ), /* (143 1) */
    scBezFactor( 0x69c6,    0.4131920 ), /* (143 2) */
    scBezFactor( 0x2c9e,    0.1742963 ), /* (143 3) */
},
#endif

#ifdef SubDiv16
{
    scBezFactor( 0x1570,    0.0837402 ), /* (144 0) */
    scBezFactor( 0x52b0,    0.3229980 ), /* (144 1) */
    scBezFactor( 0x6a50,    0.4152832 ), /* (144 2) */
    scBezFactor( 0x2d90,    0.1779785 ), /* (144 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x14de,    0.0815172 ), /* (145 0) */
    scBezFactor( 0x51c8,    0.3194591 ), /* (145 1) */
    scBezFactor( 0x6ad4,    0.4173115 ), /* (145 2) */
    scBezFactor( 0x2e84,    0.1817122 ), /* (145 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x144f,    0.0793338 ), /* (146 0) */
    scBezFactor( 0x50de,    0.3158927 ), /* (146 1) */
    scBezFactor( 0x6b55,    0.4192758 ), /* (146 2) */
    scBezFactor( 0x2f7c,    0.1854978 ), /* (146 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x13c2,    0.0771897 ), /* (147 0) */
    scBezFactor( 0x4ff2,    0.3122998 ), /* (147 1) */
    scBezFactor( 0x6bd2,    0.4211749 ), /* (147 2) */
    scBezFactor( 0x3078,    0.1893355 ), /* (147 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x1338,    0.0750847 ), /* (148 0) */
    scBezFactor( 0x4f05,    0.3086815 ), /* (148 1) */
    scBezFactor( 0x6c4a,    0.4230080 ), /* (148 2) */
    scBezFactor( 0x3177,    0.1932259 ), /* (148 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x12b1,    0.0730183 ), /* (149 0) */
    scBezFactor( 0x4e17,    0.3050389 ), /* (149 1) */
    scBezFactor( 0x6cbd,    0.4247738 ), /* (149 2) */
    scBezFactor( 0x3279,    0.1971691 ), /* (149 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x122c,    0.0709901 ), /* (150 0) */
    scBezFactor( 0x4d26,    0.3013730 ), /* (150 1) */
    scBezFactor( 0x6d2d,    0.4264712 ), /* (150 2) */
}
```

```
    scBezFactor( 0x337f,      0.200057 ) /* (150 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x11a9,      0.0689998 ), /* (151 0) */
    scBezFactor( 0x4c35,      0.2976850 ), /* (151 1) */
    scBezFactor( 0x6d97,      0.4280993 ), /* (151 2) */
    scBezFactor( 0x3489,      0.2052159 ) /* (151 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0x112a,      0.0670471 ), /* (152 0) */
    scBezFactor( 0x4b42,      0.2939758 ), /* (152 1) */
    scBezFactor( 0x6dfe,      0.4296570 ), /* (152 2) */
    scBezFactor( 0x3596,      0.2093201 ) /* (152 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x10ac,      0.0651316 ), /* (153 0) */
    scBezFactor( 0x4a4d,      0.2902467 ), /* (153 1) */
    scBezFactor( 0x6e5f,      0.4311431 ), /* (153 2) */
    scBezFactor( 0x36a6,      0.2134786 ) /* (153 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x1031,      0.0632529 ), /* (154 0) */
    scBezFactor( 0x4957,      0.2864985 ), /* (154 1) */
    scBezFactor( 0x6ebc,      0.4325566 ), /* (154 2) */
    scBezFactor( 0x37ba,      0.2176919 ) /* (154 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0fb8,      0.0614107 ), /* (155 0) */
    scBezFactor( 0x4861,      0.2827325 ), /* (155 1) */
    scBezFactor( 0x6f13,      0.4338965 ), /* (155 2) */
    scBezFactor( 0x38d2,      0.2219602 ) /* (155 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x0f42,      0.0596046 ), /* (156 0) */
    scBezFactor( 0x4769,      0.2789497 ), /* (156 1) */
    scBezFactor( 0x6f66,      0.4351616 ), /* (156 2) */
    scBezFactor( 0x39ed,      0.2262840 ) /* (156 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0ece,      0.0578343 ), /* (157 0) */
    scBezFactor( 0x4670,      0.2751512 ), /* (157 1) */
    scBezFactor( 0x6fb4,      0.4363509 ), /* (157 2) */
    scBezFactor( 0x3b0c,      0.2306636 ) /* (157 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x0e5c,    0.0560994 ), /* (158 0) */
    scBezFactor( 0x4576,    0.2713380 ), /* (158 1) */
    scBezFactor( 0x6ffd,    0.4374633 ), /* (158 2) */
    scBezFactor( 0x3c2f,    0.2350993 ), /* (158 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0ded,    0.0543995 ), /* (159 0) */
    scBezFactor( 0x447b,    0.2675112 ), /* (159 1) */
    scBezFactor( 0x7041,    0.4384977 ), /* (159 2) */
    scBezFactor( 0x3d55,    0.2395915 ), /* (159 3) */
},
#endif
```

```
#ifdef SubDiv8
{
    scBezFactor( 0x0d80,    0.0527344 ), /* (160 0) */
    scBezFactor( 0x4380,    0.2636719 ), /* (160 1) */
    scBezFactor( 0x7080,    0.4394531 ), /* (160 2) */
    scBezFactor( 0x3e80,    0.2441406 ), /* (160 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0d15,    0.0511035 ), /* (161 0) */
    scBezFactor( 0x4283,    0.2598211 ), /* (161 1) */
    scBezFactor( 0x70b9,    0.4403284 ), /* (161 2) */
    scBezFactor( 0x3fad,    0.2487469 ), /* (161 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x0cac,    0.0495067 ), /* (162 0) */
    scBezFactor( 0x4186,    0.2559600 ), /* (162 1) */
    scBezFactor( 0x70ed,    0.4411225 ), /* (162 2) */
    scBezFactor( 0x40df,    0.2534108 ), /* (162 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0c46,    0.0479434 ), /* (163 0) */
    scBezFactor( 0x4088,    0.2520896 ), /* (163 1) */
    scBezFactor( 0x711c,    0.4418344 ), /* (163 2) */
    scBezFactor( 0x4214,    0.2581326 ), /* (163 3) */
},
#endif
```

```
#ifdef SubDiv64
{
    scBezFactor( 0x0be1,    0.0464134 ), /* (164 0) */
    scBezFactor( 0x3f8a,    0.2482109 ), /* (164 1) */
    scBezFactor( 0x7145,    0.4424629 ), /* (164 2) */
    scBezFactor( 0x434e,    0.2629128 ), /* (164 3) */
},
#endif
```

```
#ifdef SubDiv256
```

```
{
    scBezFactor( 0x0b7f,    0.0449163 ),    /* (165 0) */
    scBezFactor( 0x3e8c,    0.2443251 ),    /* (165 1) */
    scBezFactor( 0x7168,    0.4430071 ),    /* (165 2) */
    scBezFactor( 0x448b,    0.2677515 ),    /* (165 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0b1f,    0.0434518 ),    /* (166 0) */
    scBezFactor( 0x3d8d,    0.2404332 ),    /* (166 1) */
    scBezFactor( 0x7186,    0.4434657 ),    /* (166 2) */
    scBezFactor( 0x45cc,    0.2726493 ),    /* (166 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0ac1,    0.0420194 ),    /* (167 0) */
    scBezFactor( 0x3c8d,    0.2365363 ),    /* (167 1) */
    scBezFactor( 0x719f,    0.4438378 ),    /* (167 2) */
    scBezFactor( 0x4711,    0.2776064 ),    /* (167 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0x0a66,    0.0406189 ),    /* (168 0) */
    scBezFactor( 0x3b8e,    0.2326355 ),    /* (168 1) */
    scBezFactor( 0x71b2,    0.4441223 ),    /* (168 2) */
    scBezFactor( 0x485a,    0.2826233 ),    /* (168 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0a0c,    0.0392498 ),    /* (169 0) */
    scBezFactor( 0x3a8e,    0.2287318 ),    /* (169 1) */
    scBezFactor( 0x71be,    0.4443181 ),    /* (169 2) */
    scBezFactor( 0x49a6,    0.2877002 ),    /* (169 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x09b4,    0.0379119 ),    /* (170 0) */
    scBezFactor( 0x398e,    0.2248263 ),    /* (170 1) */
    scBezFactor( 0x71c5,    0.4444242 ),    /* (170 2) */
    scBezFactor( 0x4af7,    0.2928376 ),    /* (170 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x095e,    0.0366047 ),    /* (171 0) */
    scBezFactor( 0x388e,    0.2209201 ),    /* (171 1) */
    scBezFactor( 0x71c6,    0.4444394 ),    /* (171 2) */
    scBezFactor( 0x4c4c,    0.2980358 ),    /* (171 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x090b,    0.0353279 ),    /* (172 0) */
    scBezFactor( 0x378e,    0.2170143 ),    /* (172 1) */
}
```

```
    scBezFactor( 0x71c1,    0.44196 ),    /* (172 2) */
    scBezFactor( 0x4da4,    0.3032951 ),    /* (172 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x08b9,    0.0340812 ),    /* (173 0) */
    scBezFactor( 0x368e,    0.2131099 ),    /* (173 1) */
    scBezFactor( 0x71b6,    0.4441929 ),    /* (173 2) */
    scBezFactor( 0x4f01,    0.3086160 ),    /* (173 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0869,    0.0328641 ),    /* (174 0) */
    scBezFactor( 0x358e,    0.2092080 ),    /* (174 1) */
    scBezFactor( 0x71a5,    0.4439292 ),    /* (174 2) */
    scBezFactor( 0x5062,    0.3139987 ),    /* (174 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x081b,    0.0316764 ),    /* (175 0) */
    scBezFactor( 0x348f,    0.2053097 ),    /* (175 1) */
    scBezFactor( 0x718d,    0.4435703 ),    /* (175 2) */
    scBezFactor( 0x51c7,    0.3194436 ),    /* (175 3) */
},
#endif

#ifdef SubDiv16
{
    scBezFactor( 0x07d0,    0.0305176 ),    /* (176 0) */
    scBezFactor( 0x3390,    0.2014160 ),    /* (176 1) */
    scBezFactor( 0x7170,    0.4431152 ),    /* (176 2) */
    scBezFactor( 0x5330,    0.3249512 ),    /* (176 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0785,    0.0293874 ),    /* (177 0) */
    scBezFactor( 0x3291,    0.1975281 ),    /* (177 1) */
    scBezFactor( 0x714b,    0.4425629 ),    /* (177 2) */
    scBezFactor( 0x549d,    0.3305216 ),    /* (177 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x073d,    0.0282855 ),    /* (178 0) */
    scBezFactor( 0x3192,    0.1936469 ),    /* (178 1) */
    scBezFactor( 0x7121,    0.4419122 ),    /* (178 2) */
    scBezFactor( 0x560e,    0.3361554 ),    /* (178 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x06f7,    0.0272115 ),    /* (179 0) */
    scBezFactor( 0x3095,    0.1897736 ),    /* (179 1) */
    scBezFactor( 0x70ef,    0.4411620 ),    /* (179 2) */
    scBezFactor( 0x5783,    0.3418528 ),    /* (179 3) */
},
#endif
```

#endif

#ifdef SubDiv64

```
{
    scBezFactor( 0x06b2,    0.0261650 ), /* (180 0) */
    scBezFactor( 0x2f97,    0.1859093 ), /* (180 1) */
    scBezFactor( 0x70b8,    0.4403114 ), /* (180 2) */
    scBezFactor( 0x58fd,    0.3476143 ) /* (180 3) */
},
#endif
```

#ifdef SubDiv256

```
{
    scBezFactor( 0x066f,    0.0251457 ), /* (181 0) */
    scBezFactor( 0x2e9b,    0.1820549 ), /* (181 1) */
    scBezFactor( 0x7079,    0.4393592 ), /* (181 2) */
    scBezFactor( 0x5a7b,    0.3534401 ) /* (181 3) */
},
#endif
```

#ifdef SubDiv128

```
{
    scBezFactor( 0x062e,    0.0241532 ), /* (182 0) */
    scBezFactor( 0x2d9f,    0.1782117 ), /* (182 1) */
    scBezFactor( 0x7034,    0.4383044 ), /* (182 2) */
    scBezFactor( 0x5bfd,    0.3593307 ) /* (182 3) */
},
#endif
```

#ifdef SubDiv256

```
{
    scBezFactor( 0x05ef,    0.0231872 ), /* (183 0) */
    scBezFactor( 0x2ca4,    0.1743806 ), /* (183 1) */
    scBezFactor( 0x6fe8,    0.4371459 ), /* (183 2) */
    scBezFactor( 0x5d83,    0.3652863 ) /* (183 3) */
},
#endif
```

#ifdef SubDiv32

```
{
    scBezFactor( 0x05b2,    0.0222473 ), /* (184 0) */
    scBezFactor( 0x2baa,    0.1705627 ), /* (184 1) */
    scBezFactor( 0x6f96,    0.4358826 ), /* (184 2) */
    scBezFactor( 0x5f0e,    0.3713074 ) /* (184 3) */
},
#endif
```

#ifdef SubDiv256

```
{
    scBezFactor( 0x0576,    0.0213332 ), /* (185 0) */
    scBezFactor( 0x2ab0,    0.1667592 ), /* (185 1) */
    scBezFactor( 0x6f3c,    0.4345134 ), /* (185 2) */
    scBezFactor( 0x609c,    0.3773943 ) /* (185 3) */
},
#endif
```

#ifdef SubDiv128

```
{
    scBezFactor( 0x053b,    0.0204444 ), /* (186 0) */
    scBezFactor( 0x29b8,    0.1629710 ), /* (186 1) */
    scBezFactor( 0x6edb,    0.4330373 ), /* (186 2) */
    scBezFactor( 0x6230,    0.3835473 ) /* (186 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0503,    0.0195807 ), /* (187 0) */
    scBezFactor( 0x28c1,    0.1591993 ), /* (187 1) */
    scBezFactor( 0x6e73,    0.4314532 ), /* (187 2) */
    scBezFactor( 0x63c7,    0.3897669 ), /* (187 3) */
},
#endif
```

```
#ifdef SubDiv64
{
    scBezFactor( 0x04cc,    0.0187416 ), /* (188 0) */
    scBezFactor( 0x27cb,    0.1554451 ), /* (188 1) */
    scBezFactor( 0x6e04,    0.4297600 ), /* (188 2) */
    scBezFactor( 0x6563,    0.3960533 ), /* (188 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0496,    0.0179269 ), /* (189 0) */
    scBezFactor( 0x26d6,    0.1517095 ), /* (189 1) */
    scBezFactor( 0x6d8e,    0.4279566 ), /* (189 2) */
    scBezFactor( 0x6704,    0.4024070 ), /* (189 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x0463,    0.0171361 ), /* (190 0) */
    scBezFactor( 0x25e2,    0.1479936 ), /* (190 1) */
    scBezFactor( 0x6d11,    0.4260421 ), /* (190 2) */
    scBezFactor( 0x68a8,    0.4088283 ), /* (190 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0430,    0.0163689 ), /* (191 0) */
    scBezFactor( 0x24f0,    0.1442984 ), /* (191 1) */
    scBezFactor( 0x6c8c,    0.4240152 ), /* (191 2) */
    scBezFactor( 0x6a52,    0.4153175 ), /* (191 3) */
},
#endif
```

```
#ifdef SubDiv4
{
    scBezFactor( 0x0400,    0.0156250 ), /* (192 0) */
    scBezFactor( 0x2400,    0.1406250 ), /* (192 1) */
    scBezFactor( 0x6c00,    0.4218750 ), /* (192 2) */
    scBezFactor( 0x6c00,    0.4218750 ), /* (192 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x03d0,    0.0149040 ), /* (193 0) */
    scBezFactor( 0x2310,    0.1369745 ), /* (193 1) */
    scBezFactor( 0x6b6c,    0.4196203 ), /* (193 2) */
    scBezFactor( 0x6db2,    0.4285012 ), /* (193 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x03a2,    0.0142055 ), /* (194 0) */

```



```
    scBezFactor( 0x2223,      0.133502 ), /* (194 1) */
    scBezFactor( 0x6ad0,      0.4172502 ), /* (194 2) */
    scBezFactor( 0x6f69,      0.4351964 ) /* (194 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0376,      0.0135291 ), /* (195 0) */
    scBezFactor( 0x2137,      0.1297465 ), /* (195 1) */
    scBezFactor( 0x6a2d,      0.4147634 ), /* (195 2) */
    scBezFactor( 0x7124,      0.4419610 ) /* (195 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x034b,      0.0128746 ), /* (196 0) */
    scBezFactor( 0x204c,      0.1261711 ), /* (196 1) */
    scBezFactor( 0x6983,      0.4121590 ), /* (196 2) */
    scBezFactor( 0x72e4,      0.4487953 ) /* (196 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0322,      0.0122415 ), /* (197 0) */
    scBezFactor( 0x1f64,      0.1226229 ), /* (197 1) */
    scBezFactor( 0x68d0,      0.4094358 ), /* (197 2) */
    scBezFactor( 0x74a8,      0.4556997 ) /* (197 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x02fa,      0.0116296 ), /* (198 0) */
    scBezFactor( 0x1e7d,      0.1191030 ), /* (198 1) */
    scBezFactor( 0x6816,      0.4065928 ), /* (198 2) */
    scBezFactor( 0x7671,      0.4626746 ) /* (198 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x02d3,      0.0110384 ), /* (199 0) */
    scBezFactor( 0x1d98,      0.1156123 ), /* (199 1) */
    scBezFactor( 0x6754,      0.4036290 ), /* (199 2) */
    scBezFactor( 0x783f,      0.4697203 ) /* (199 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0x02ae,      0.0104675 ), /* (200 0) */
    scBezFactor( 0x1cb6,      0.1121521 ), /* (200 1) */
    scBezFactor( 0x668a,      0.4005432 ), /* (200 2) */
    scBezFactor( 0x7a12,      0.4768372 ) /* (200 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0289,      0.0099167 ), /* (201 0) */
    scBezFactor( 0x1bd5,      0.1087233 ), /* (201 1) */
    scBezFactor( 0x65b7,      0.3973344 ), /* (201 2) */
    scBezFactor( 0x7be9,      0.4840255 ) /* (201 3) */
}
```

```
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0267, 0.0093856 ), /* (202 0) */
    scBezFactor( 0x1af6, 0.1053271 ), /* (202 1) */
    scBezFactor( 0x64dd, 0.3940015 ), /* (202 2) */
    scBezFactor( 0x7dc4, 0.4912858 ), /* (202 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0245, 0.0088738 ), /* (203 0) */
    scBezFactor( 0x1a1a, 0.1019645 ), /* (203 1) */
    scBezFactor( 0x63fa, 0.3905434 ), /* (203 2) */
    scBezFactor( 0x7fa5, 0.4986183 ), /* (203 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x0225, 0.0083809 ), /* (204 0) */
    scBezFactor( 0x1940, 0.0986366 ), /* (204 1) */
    scBezFactor( 0x630f, 0.3869591 ), /* (204 2) */
    scBezFactor( 0x818a, 0.5060234 ), /* (204 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0206, 0.0079066 ), /* (205 0) */
    scBezFactor( 0x1868, 0.0953445 ), /* (205 1) */
    scBezFactor( 0x621c, 0.3832474 ), /* (205 2) */
    scBezFactor( 0x8374, 0.5135015 ), /* (205 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x01e8, 0.0074506 ), /* (206 0) */
    scBezFactor( 0x1793, 0.0920892 ), /* (206 1) */
    scBezFactor( 0x6120, 0.3794074 ), /* (206 2) */
    scBezFactor( 0x8563, 0.5210528 ), /* (206 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x01cb, 0.0070124 ), /* (207 0) */
    scBezFactor( 0x16c0, 0.0888718 ), /* (207 1) */
    scBezFactor( 0x601c, 0.3754379 ), /* (207 2) */
    scBezFactor( 0x8757, 0.5286779 ), /* (207 3) */
},
#endif

#ifdef SubDiv16
{
    scBezFactor( 0x01b0, 0.0065918 ), /* (208 0) */
    scBezFactor( 0x15f0, 0.0856934 ), /* (208 1) */
    scBezFactor( 0x5f10, 0.3713379 ), /* (208 2) */
    scBezFactor( 0x8950, 0.5363770 ), /* (208 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0195,    0.0061883 ), /* (209 0) */
    scBezFactor( 0x1522,    0.0825550 ), /* (209 1) */
    scBezFactor( 0x5dfa,    0.3671063 ), /* (209 2) */
    scBezFactor( 0x8b4d,    0.5441504 ), /* (209 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x017c,    0.0058017 ), /* (210 0) */
    scBezFactor( 0x1457,    0.0794578 ), /* (210 1) */
    scBezFactor( 0x5cdc,    0.3627419 ), /* (210 2) */
    scBezFactor( 0x8d4f,    0.5519986 ), /* (210 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0163,    0.0054315 ), /* (211 0) */
    scBezFactor( 0x138f,    0.0764027 ), /* (211 1) */
    scBezFactor( 0x5bb5,    0.3582439 ), /* (211 2) */
    scBezFactor( 0x8f57,    0.5599219 ), /* (211 3) */
},
#endif
```

```
#ifdef SubDiv64
{
    scBezFactor( 0x014c,    0.0050774 ), /* (212 0) */
    scBezFactor( 0x12c9,    0.0733910 ), /* (212 1) */
    scBezFactor( 0x5a86,    0.3536110 ), /* (212 2) */
    scBezFactor( 0x9163,    0.5679207 ), /* (212 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x0136,    0.0047390 ), /* (213 0) */
    scBezFactor( 0x1207,    0.0704235 ), /* (213 1) */
    scBezFactor( 0x594d,    0.3488422 ), /* (213 2) */
    scBezFactor( 0x9374,    0.5759953 ), /* (213 3) */
},
#endif
```

```
#ifdef SubDiv128
{
    scBezFactor( 0x0121,    0.0044160 ), /* (214 0) */
    scBezFactor( 0x1147,    0.0675015 ), /* (214 1) */
    scBezFactor( 0x580c,    0.3439364 ), /* (214 2) */
    scBezFactor( 0x958a,    0.5841460 ), /* (214 3) */
},
#endif
```

```
#ifdef SubDiv256
{
    scBezFactor( 0x010d,    0.0041080 ), /* (215 0) */
    scBezFactor( 0x108b,    0.0646260 ), /* (215 1) */
    scBezFactor( 0x56c1,    0.3388926 ), /* (215 2) */
    scBezFactor( 0x97a5,    0.5923733 ), /* (215 3) */
},
#endif
```

```
#ifdef SubDiv32
{
```

```
    scBezFactor( 0x00fa,    0.0017981 ), /* (216 0) */
    scBezFactor( 0x0fd2,    0.0617981 ), /* (216 1) */
    scBezFactor( 0x556e,    0.3337097 ), /* (216 2) */
    scBezFactor( 0x99c6,    0.6006775 ) /* (216 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x00e7,    0.0035357 ), /* (217 0) */
    scBezFactor( 0x0f1b,    0.0590188 ), /* (217 1) */
    scBezFactor( 0x5411,    0.3283866 ), /* (217 2) */
    scBezFactor( 0x9beb,    0.6090589 ) /* (217 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x00d6,    0.0032706 ), /* (218 0) */
    scBezFactor( 0x0e68,    0.0562892 ), /* (218 1) */
    scBezFactor( 0x52ab,    0.3229222 ), /* (218 2) */
    scBezFactor( 0x9e15,    0.6175179 ) /* (218 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x00c5,    0.0030192 ), /* (219 0) */
    scBezFactor( 0x0db9,    0.0536104 ), /* (219 1) */
    scBezFactor( 0x513b,    0.3173155 ), /* (219 2) */
    scBezFactor( 0xa045,    0.6260549 ) /* (219 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x00b6,    0.0027809 ), /* (220 0) */
    scBezFactor( 0x0d0d,    0.0509834 ), /* (220 1) */
    scBezFactor( 0x4fc2,    0.3115654 ), /* (220 2) */
    scBezFactor( 0xa279,    0.6346703 ) /* (220 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x00a7,    0.0025555 ), /* (221 0) */
    scBezFactor( 0x0c64,    0.0484094 ), /* (221 1) */
    scBezFactor( 0x4e40,    0.3056708 ), /* (221 2) */
    scBezFactor( 0xa4b3,    0.6433643 ) /* (221 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0099,    0.0023427 ), /* (222 0) */
    scBezFactor( 0x0bbf,    0.0458894 ), /* (222 1) */
    scBezFactor( 0x4cb4,    0.2996306 ), /* (222 2) */
    scBezFactor( 0xa6f2,    0.6521373 ) /* (222 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x008c,    0.0021420 ), /* (223 0) */
    scBezFactor( 0x0b1d,    0.0434244 ), /* (223 1) */
    scBezFactor( 0x4b1f,    0.2934439 ) /* (223 2) */
}
```

```

    scBezFactor( 0xa936,      0.660000 ) /* (223 3) */
},
#endif

#ifdef SubDiv8
{
    scBezFactor( 0x0080,      0.0019531 ), /* (224 0) */
    scBezFactor( 0x0a80,      0.0410156 ), /* (224 1) */
    scBezFactor( 0x4980,      0.2871094 ), /* (224 2) */
    scBezFactor( 0xab80,      0.6699219 ), /* (224 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0074,      0.0017757 ), /* (225 0) */
    scBezFactor( 0x09e5,      0.0386640 ), /* (225 1) */
    scBezFactor( 0x47d7,      0.2806261 ), /* (225 2) */
    scBezFactor( 0xadce,      0.6789342 ), /* (225 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0069,      0.0016093 ), /* (226 0) */
    scBezFactor( 0x094f,      0.0363708 ), /* (226 1) */
    scBezFactor( 0x4624,      0.2739930 ), /* (226 2) */
    scBezFactor( 0xb022,      0.6880269 ), /* (226 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x005f,      0.0014537 ), /* (227 0) */
    scBezFactor( 0x08bd,      0.0341368 ), /* (227 1) */
    scBezFactor( 0x4467,      0.2672090 ), /* (227 2) */
    scBezFactor( 0xb27b,      0.6972005 ), /* (227 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x0055,      0.0013084 ), /* (228 0) */
    scBezFactor( 0x082e,      0.0319633 ), /* (228 1) */
    scBezFactor( 0x42a1,      0.2602730 ), /* (228 2) */
    scBezFactor( 0xb4da,      0.7064552 ), /* (228 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x004c,      0.0011732 ), /* (229 0) */
    scBezFactor( 0x07a4,      0.0298514 ), /* (229 1) */
    scBezFactor( 0x40d0,      0.2531839 ), /* (229 2) */
    scBezFactor( 0xb73e,      0.7157915 ), /* (229 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0044,      0.0010476 ), /* (230 0) */
    scBezFactor( 0x071e,      0.0278020 ), /* (230 1) */
    scBezFactor( 0x3ef5,      0.2459407 ), /* (230 2) */
    scBezFactor( 0xb9a7,      0.7252097 ), /* (230 3) */
},
#endif

```

```
#ifdef SubDiv256
{
    scBezFactor( 0x003d,    0.0009313 ), /* (231 0) */
    scBezFactor( 0x069b,    0.0258163 ), /* (231 1) */
    scBezFactor( 0x3d11,    0.2385423 ), /* (231 2) */
    scBezFactor( 0xbc15,    0.7347102 ) /* (231 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0x0036,    0.0008240 ), /* (232 0) */
    scBezFactor( 0x061e,    0.0238953 ), /* (232 1) */
    scBezFactor( 0x3b22,    0.2309875 ), /* (232 2) */
    scBezFactor( 0xbe8a,    0.7442932 ) /* (232 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x002f,    0.0007252 ), /* (233 0) */
    scBezFactor( 0x05a4,    0.0220401 ), /* (233 1) */
    scBezFactor( 0x3928,    0.2232755 ), /* (233 2) */
    scBezFactor( 0xc103,    0.7539592 ) /* (233 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0029,    0.0006347 ), /* (234 0) */
    scBezFactor( 0x052f,    0.0202518 ), /* (234 1) */
    scBezFactor( 0x3724,    0.2154050 ), /* (234 2) */
    scBezFactor( 0xc382,    0.7637086 ) /* (234 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0024,    0.0005520 ), /* (235 0) */
    scBezFactor( 0x04be,    0.0185314 ), /* (235 1) */
    scBezFactor( 0x3516,    0.2073750 ), /* (235 2) */
    scBezFactor( 0xc606,    0.7735416 ) /* (235 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x001f,    0.0004768 ), /* (236 0) */
    scBezFactor( 0x0452,    0.0168800 ), /* (236 1) */
    scBezFactor( 0x32fd,    0.1991844 ), /* (236 2) */
    scBezFactor( 0xc890,    0.7834587 ) /* (236 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x001a,    0.0004088 ), /* (237 0) */
    scBezFactor( 0x03ea,    0.0152988 ), /* (237 1) */
    scBezFactor( 0x30da,    0.1908322 ), /* (237 2) */
    scBezFactor( 0xcb20,    0.7934602 ) /* (237 3) */
},
#endif

#ifdef SubDiv128
```

```
{
    scBezFactor( 0x0016,      0.0003476 ), /* (238 0) */
    scBezFactor( 0x0387,      0.0137887 ), /* (238 1) */
    scBezFactor( 0x2eac,      0.1823173 ), /* (238 2) */
    scBezFactor( 0xcd5,       0.8035464 ), /* (238 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0013,      0.0002928 ), /* (239 0) */
    scBezFactor( 0x0329,      0.0123509 ), /* (239 1) */
    scBezFactor( 0x2c73,      0.1736385 ), /* (239 2) */
    scBezFactor( 0xd04f,      0.8137178 ), /* (239 3) */
},
#endif

#ifdef SubDiv16
{
    scBezFactor( 0x0010,      0.0002441 ), /* (240 0) */
    scBezFactor( 0x02d0,      0.0109863 ), /* (240 1) */
    scBezFactor( 0x2a30,      0.1647949 ), /* (240 2) */
    scBezFactor( 0xd2f0,      0.8239746 ), /* (240 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x000d,      0.0002012 ), /* (241 0) */
    scBezFactor( 0x027b,      0.0096962 ), /* (241 1) */
    scBezFactor( 0x27e1,      0.1557854 ), /* (241 2) */
    scBezFactor( 0xd595,      0.8343173 ), /* (241 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x000a,      0.0001636 ), /* (242 0) */
    scBezFactor( 0x022b,      0.0084815 ), /* (242 1) */
    scBezFactor( 0x2588,      0.1466088 ), /* (242 2) */
    scBezFactor( 0xd841,      0.8447461 ), /* (242 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0008,      0.0001310 ), /* (243 0) */
    scBezFactor( 0x01e1,      0.0073434 ), /* (243 1) */
    scBezFactor( 0x2323,      0.1372642 ), /* (243 2) */
    scBezFactor( 0xdaf2,      0.8552615 ), /* (243 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x0006,      0.0001030 ), /* (244 0) */
    scBezFactor( 0x019b,      0.0062828 ), /* (244 1) */
    scBezFactor( 0x20b4,      0.1277504 ), /* (244 2) */
    scBezFactor( 0xdda9,      0.8658638 ), /* (244 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0005,      0.0000793 ), /* (245 0) */
    scBezFactor( 0x015b,      0.0053009 ), /* (245 1) */
}
```

```

    scBezFactor( 0x1e39,      0.118884 ), /* (245 2) */
    scBezFactor( 0xe065,      0.8765534 ) /* (245 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0003,      0.0000596 ), /* (246 0) */
    scBezFactor( 0x0120,      0.0043988 ), /* (246 1) */
    scBezFactor( 0x1bb3,      0.1082110 ), /* (246 2) */
    scBezFactor( 0xe328,      0.8873305 ) /* (246 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0002,      0.0000435 ), /* (247 0) */
    scBezFactor( 0x00ea,      0.0035775 ), /* (247 1) */
    scBezFactor( 0x1922,      0.0981833 ), /* (247 2) */
    scBezFactor( 0xe5f0,      0.8981957 ) /* (247 3) */
},
#endif

#ifdef SubDiv32
{
    scBezFactor( 0x0002,      0.0000305 ), /* (248 0) */
    scBezFactor( 0x00ba,      0.0028381 ), /* (248 1) */
    scBezFactor( 0x1686,      0.0879822 ), /* (248 2) */
    scBezFactor( 0xe8be,      0.9091492 ) /* (248 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0001,      0.0000204 ), /* (249 0) */
    scBezFactor( 0x008e,      0.0021817 ), /* (249 1) */
    scBezFactor( 0x13de,      0.0776065 ), /* (249 2) */
    scBezFactor( 0xeb91,      0.9201913 ) /* (249 3) */
},
#endif

#ifdef SubDiv128
{
    scBezFactor( 0x0000,      0.0000129 ), /* (250 0) */
    scBezFactor( 0x0069,      0.0016093 ), /* (250 1) */
    scBezFactor( 0x112a,      0.0670552 ), /* (250 2) */
    scBezFactor( 0xee6b,      0.9313226 ) /* (250 3) */
},
#endif

#ifdef SubDiv256
{
    scBezFactor( 0x0000,      0.0000075 ), /* (251 0) */
    scBezFactor( 0x0049,      0.0011221 ), /* (251 1) */
    scBezFactor( 0x0e6b,      0.0563273 ), /* (251 2) */
    scBezFactor( 0xf14a,      0.9425432 ) /* (251 3) */
},
#endif

#ifdef SubDiv64
{
    scBezFactor( 0x0000,      0.0000038 ), /* (252 0) */
    scBezFactor( 0x002f,      0.0007210 ), /* (252 1) */
    scBezFactor( 0x0ba0,      0.0454216 ), /* (252 2) */
    scBezFactor( 0xf42f,      0.9538536 ) /* (252 3) */
},

```


#endif

#ifdef SubDiv256

```
{
    scBezFactor( 0x0000,    0.00000016 ), /* (253 0) */
    scBezFactor( 0x001a,    0.0004072 ), /* (253 1) */
    scBezFactor( 0x08ca,    0.0343371 ), /* (253 2) */
    scBezFactor( 0xf71a,    0.9652541 ), /* (253 3) */
},
#endif
```

#ifdef SubDiv128

```
{
    scBezFactor( 0x0000,    0.00000005 ), /* (254 0) */
    scBezFactor( 0x000b,    0.0001817 ), /* (254 1) */
    scBezFactor( 0x05e8,    0.0230727 ), /* (254 2) */
    scBezFactor( 0xfa0b,    0.9767451 ), /* (254 3) */
},
#endif
```

#ifdef SubDiv256

```
{
    scBezFactor( 0x0000,    0.00000001 ), /* (255 0) */
    scBezFactor( 0x0002,    0.0000456 ), /* (255 1) */
    scBezFactor( 0x02fa,    0.0116274 ), /* (255 2) */
    scBezFactor( 0xfd02,    0.9883270 ), /* (255 3) */
},
#endif
```

/* This one is needed by all sub divisions */

```
{
    scBezFactor( 0x0000,    0.00000000 ), /* (256 0) */
    scBezFactor( 0x0000,    0.00000000 ), /* (256 1) */
    scBezFactor( 0x0000,    0.00000000 ), /* (256 2) */
    scBezFactor( 0x0000,    1.00000000 ), /* (256 3) */
},
```

File: SCBEZIER.C

\$Header: /Projects/Toolbox/ct/SCBEZIER.CPP 2 5/30/97 8:45a Wmanis \$

Contains: vectorizes beziers

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/

#include "scbezier.h"
#include "scmem.h"
#include <limits.h>

struct SCBezVertex {
    short x;
    short y;
};

/* ----- */
extern scBezBlendValue bezblend[];

static void BezCompute( scVertex* dstV,
                       const scVertex* srcV );

/* ----- */
/* count the number of vertices in the vertex list */
inline long CountVerts( const scVertex* verts )
{
    long numVerts;
    for ( numVerts = 1; verts->fPointType != eFinalPoint; verts++, numVerts++ )
        ;
    return numVerts;
}

/* ----- */
/* count the number of bezier curves in a vertex list */
inline long CountBezCurves( const scVertex* verts )
{
    long numCurves;

    for ( numCurves = 0; verts->fPointType != eFinalPoint; ) {
        if ( verts->fPointType == eBezControlPoint && (verts+1)->fPointType == eBezControlPoint ) {
            numCurves++;
            verts += 2;
        }
        else
            verts++;
    }
    return numCurves;
}

/* ----- */
/* process the list vectorizing the beziers into straight lines */

```

```

void BEZVectorizePoly( scVertex*& dstV,
                      const scVertex* srcV )
{
    scVertex*   vList;
    int         i;
    Bool        process;
    long        numPoints;
    long        numCurves;
    scVertex    bezVectors[ scBezBlendSize + 2 ];

    numPoints = CountVerts( srcV );
    numCurves = CountBezCurves( srcV );

    if ( numCurves == 0 )
        return;

    long        segments = numPoints + scBezBlendSize * numCurves;
    dstV = vList = (scVertex*)MEMAllocPtr( segments * sizeof( scVertex ) );

    for ( process = true; process; ) {
        switch ( srcV->fPointType ) {
            case eFinalPoint:
                process = false;
            default:
                *vList++ = *srcV++;
                break;

            case eBezControlPoint:
                // insure we have two points - if not process normally
                if ( (srcV+1)->fPointType == eBezControlPoint ) {
                    BezCompute( bezVectors, srcV - 1 );
                    for ( i = 1; i < scBezBlendSize - 1; i++ )
                        *vList++ = bezVectors[i];
                    srcV += 2;
                }
                else
                    *vList++ = *srcV++;
                break;
        }
    }

    /* ===== */
#ifdef scBezFixed
    /* it is assumed that the list has enough space before entering
     * this routine
     */

    static void RenderBezier( SCVertex*   dstV,
                             scPointType  type,
                             SCBezVertex* draw,
                             short         minX,
                             short         minY )
    {
        size_t i;

        for ( i = 0; i < scBezBlendSize; i++, draw++ ) {
            if ( i == 0 )
                dstV->fPointType = type;
            else
                dstV->fPointType = eCornerPoint;

            dstV->x = (long)( draw->x - minX ) << 16;
            dstV->y = (long)( draw->y - minY ) << 16;
            dstV++;
        }
    }
}

```

```

/* -----
SCBezVertex *SCBezCompDrawList( SCBezVertex*   theVerts,
                                SCBezVertex*   drawList )
{
    SCBezVertex*   pDraw;
    scBezBlendValue* pBlend;
    short          i;

    drawList[0]      = theVerts[0];
    drawList[scBezBlendSize-1] = theVerts[3];

    pBlend = bezblend + 1;
    pDraw  = drawList + 1;

    for (i = 0; i < scBezBlendSize-2; i++) {
        pDraw->x = (short)((long)theVerts[0].x * (ulong)pBlend->ca
            + (long)theVerts[1].x * (ulong)pBlend->cb
            + (long)theVerts[2].x * (ulong)pBlend->cc
            + (long)theVerts[3].x * (ulong)pBlend->cd) >> 16);
        pDraw->y = (short)((long)theVerts[0].y * (ulong)pBlend->ca
            + (long)theVerts[1].y * (ulong)pBlend->cb
            + (long)theVerts[2].y * (ulong)pBlend->cc
            + (long)theVerts[3].y * (ulong)pBlend->cd) >> 16);
        pBlend++;
        pDraw++;
    }
    return drawList;
}
/* ----- */

static void BezCompute( SCVertex*   dstV,
                       const SCVertex* srcV )
{
    SCBezVertex   v[4];
    SCBezVertex   drawList[scBezBlendSize];
    register int   i;
    scPointType   fPointType;
    short         minX;
    short         minY;

    fPointType = srcV->fPointType;

    /* put source into a 16 bit quantity so that
     * we can perform fixed point multiplies on it,
     * and force bezier into positive coordinate space
     * so that the fixed point multiplies with blending
     * values will work
     */

    minX = minY = SHRT_MAX;
    for ( i = 0; i < 4; i++ ) {
        v[i].x = (short)( srcV->x >> 16 );
        minX  = MIN( minX, v[i].x );
        v[i].y = (short)( srcV->y >> 16 );
        minY  = MIN( minY, v[i].y );
        srcV++;
    }

    minX = ( minX < 0 ? -minX : 0 );
    minY = ( minY < 0 ? -minY : 0 );
    if ( minX || minY ) {
        for ( i = 0; i < 4; i++ ) {
            v[i].x += minX;
            v[i].y += minY;
        }
    }
    SCBezCompDrawList( v, drawList );
    RenderBezier( dstV, pointType, drawList, minX, minY );
}

```

#endif

/* ===== */

#ifdef scBezREAL

```

static void BezCompute( scVertex*      dstV,
                       const scVertex* srcV )
{
    int          i;

    dstV[0]      = srcV[0];
    dstV[scBezBlendSize-1] = srcV[3];

    for ( i = 1; i < scBezBlendSize - 1; i++ ) {
        dstV[i].x = scRoundMP( srcV[0].x * bezblend[i].ca +
                               srcV[1].x * bezblend[i].cb +
                               srcV[2].x * bezblend[i].cc +
                               srcV[3].x * bezblend[i].cd );

        dstV[i].y = scRoundMP( srcV[0].y * bezblend[i].ca +
                               srcV[1].y * bezblend[i].cb +
                               srcV[2].y * bezblend[i].cc +
                               srcV[3].y * bezblend[i].cd );

        dstV[i].fPointType = eCornerPoint;
    }
}

```

#endif

/* ===== */

File: SCBEZIER.H

\$Header: /Projects/Toolbox/ct/SCBEZIER.H 2 5/30/97 8:44a Wmanis \$

Contains: size of bezier sub-division factors

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/
#ifndef _H_SCBEZIER
#define _H_SCBEZIER

#include "sctypes.h"

void BEZVectorizePoly( scVertex *&, const scVertex * );

/*
 * At least one of these defined determines the number of vectors
 * that will be created by sub-dividing the bezier curver
 */
#define SubDiv2 2
#define SubDiv4 (SubDiv2 * 2)
#define SubDiv8 (SubDiv4 * 2)
#define SubDiv16 (SubDiv8 * 2)

#if 0
#define SubDiv32 (SubDiv16 * 2)
#define SubDiv64 (SubDiv32 * 2)
#define SubDiv128 (SubDiv64 * 2)
#define SubDiv256 (SubDiv128 * 2)
#endif

// #define scBezFixed
#define scBezREAL

#ifdef scBezFixed
#define scBezFactor( x, y ) x
struct scBezBlendValue {
    ushort ca;
    ushort cb;
    ushort cc;
    ushort cd;
};
#ifdef scBezREAL
#error "can't define both"
#endif
#endif

#ifdef scBezREAL
#define scBezFactor( x, y ) y
struct scBezBlendValue {
    REAL ca;
    REAL cb;
    REAL cc;
    REAL cd;
};
#ifdef scBezFixed
#error "can't define both"

```

[illegible]

File: SCBREAK.C

\$Header: /Projects/Toolbox/ct/Scbreak.cpp 6 5/30/97 8:45a Wmanis \$

Contains: line breaker

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/

#include "scbreak.h"
#include "sccolumn.h"
#include "scglobda.h"
#include "scstcach.h"
#include "scctype.h"
#include "scmem.h"
#include "scrfdat.h"
#include "sccallbk.h"

// TOOLBOX BEHAVIOR - force first word on line to break if it does not fit
#define scForceBreakFirstWord

#define MAXLEADVALS      50
#define MAXBREAKVALS    100

#ifndef LETTERSPACE
#define LETTERSPACE( ch ) ( (ch)->character!=scWordSpace\
                           && ((ch)+1)->character!=scWordSpace )
#endif

/* ***** */
/* ***** */
/* lint -esym(534,BRKLineDecision) */

static MicroPoint  BRKNextTokenWidth( CharRecordP, UCS2 );
static Bool        TabBreakChar( UCS2 theCh, UCS2 breakCh );

static eBreakEvent  BRKLoopBody( void );
static eBreakEvent  BRKTheLoop( void );

static Bool        BRKStillMoreChars( CharRecordP, long );

static void         BRKCharJapanese( void );

static eBreakEvent  bmBRKWordSpace( void );
static eBreakEvent  bmBRKFixSpace( void );
static eBreakEvent  bmBRKRelSpace( void );
static eBreakEvent  bmBRKEndStream( void );
static eBreakEvent  bmBRKChar( void );
static eBreakEvent  bmBRKHardReturn( void );
static eBreakEvent  bmBRKQuad( void );
static eBreakEvent  bmBRKField( void );
static eBreakEvent  bmBRKVertTab( void );
static eBreakEvent  bmBRKTab( void );
static eBreakEvent  bmBRKFillSpace( void );
static eBreakEvent  bmBRKRule( void );
static eBreakEvent  bmBRKHyphen( void );

```



```

static void      BRKDropCapControl( MicroPoint, MicroPoint );
static void      BRKPlaceLine( scMuPoint&, MicroPoint&, const scFlowDir& );
static void      BRKJustifyLine( void );
static void      BRKSpecial( ushort );
static MicroPoint BRKRagControl( CharRecordP,
                                MicroPoint, MicroPoint,
                                MicroPoint, TypeSpec, ushort, short );

static CandBreak* BRKLineDecisionJust( void );
static CandBreak* BRKLineDecisionRag( void );
static void      BRKSetCharIndent( CharRecordP, long, long, MicroPoint );

static CandBreak* BRKHyphenateRag( void );
static CandBreak* BRKHyphenateJust( void );

static void      BRKAddHyphens( CandBreak*, CandBreak* );
#ifdef scForceBreakFirstWord
static void      BRKForceHyphens( CandBreak*, CandBreak* );
#endif
static short     BRKPerformDiscHyphen( CharRecordP, CharRecordP, Hyphen* );

static void      BRKAdjustWordSpace( CharRecordP, GlyphSize, long, long );

static void      BRKRepairLastSpace( CharRecordP, long );
static void      BRKRepairFinalSpace( void );
static CharRecordP BRKLastCharOnLine( CharRecordP );
static MicroPoint BRKHangPuncRightAdjust( void );

static TypeSpec  BRKUpdateSpec( scSpecRecord* );

static MicroPoint BRKKernCorrection( CharRecordP );

static void      BRKMaxLineVals( scLINERefData&, MicroPoint, eFntBaseline, MicroPoint );

/* @@@@@@@@@@ */
/* *****/
CandBreak::CandBreak()
{
    Init();
}

/* *****/
void CandBreak::Init()
{
    breakCount      = 0;
    startCount      = 0;
    streamCount      = 0;
    wsSpaceCount     = 0;
    spaceCount       = 0;
    trailingSpaces   = 0;
    chCount          = 0;
    fillSpCount      = 0;
    lineVal          = 0;
    breakVal         = eUndefinedBreak;
    minGlue          = 0;
    optGlue          = 0;
    maxGlue          = 0;
    curBox           = 0;
    fHangable        = 0;
    theChRec         = 0;
    specChanged      = 0;
    spec.clear();
    specRec          = 0;
}

/* *****/
// if the break candidates fill we remove the first entry and shuffle the
// candidates down since the first candidate is no longer a real candidate

```

```

static void ShuffleBreakCandidates
{
    gbrS.candBreak[0].Init(); // force the clearing of a spec

    // shuffle the array down into the spot we just cleared
    SCmemmove( gbrS.candBreak, gbrS.candBreak+1, sizeof(CandBreak) * (MAXBREAKVALS-1L) );

    // zero out the last entry which is now doubled because we copied it into
    // the next to last entry
    SCmemset( gbrS.candBreak + ( MAXBREAKVALS - 1 ), 0, sizeof(CandBreak) );

    // initialize the last entry
    gbrS.candBreak[MAXBREAKVALS-1].Init();
}

/*****
/* put the current break values into the candidate break array */
static void BRKSetCandBreak( eBreakType breakType )
{
    CandBreak      *theBreak  = gbrS.candBreak + gbrS.cb.breakCount;

    *theBreak      = gbrS.cb;
    theBreak->breakVal = breakType;

    if ( breakType == eHyphBreak )
        theBreak->curBox += scCachedStyle::GetCurrentCache().GetEscapement( '-' );

    if ( gbrS.cb.breakCount >= (MAXBREAKVALS-1) )
        ShuffleBreakCandidates();
    else
        gbrS.cb.breakCount++;
}

/*****
static eBreakEvent BRKExitLoop( )
{
    return measure_exceeded;
}

/*****
/* we are passed in some characters & line attributes, break the line,
/* performing justification, hyphenation, indents, etc.
*/
eBreakType
BRKRomanLineBreak( CharRecordP      chRec,      // the character array
                  long              startCount, // # into char array to start the linebreak
                  long&             count,      // count into char array of end of line
                  scLINERefData&    lineData,
                  short             lineCount,
                  short&            linesHyphenated,
                  scSpecRecord**    specRec,
                  scXRect&          letterSpace )
{
    /* not a good idea to use register or auto variables that are used
    * across a call to setjmp/longjmp, they alter them when the previous
    * stack is restored
    */
    MicroPoint      initialLead;
    eFntBaseline     initialBaseline;

    gbrS.Init();

    /* set up state variables */
    gbrS.gStartRec = gbrS.cb.theChRec = chRec + startCount;

    gbrS.fMaxLineVals[0] = gbrS.fZeroMaxLineVals;

    lineData.fInkExtents.Translate( lineData.fOrg );

```

```

    /* zero out the char count at the start of every paragraph
    if ( startCount == 0 )
        gbrS.charIndent = LONG_MIN;
    gbrS.foundCharIndent = false;

    gbrS.cB.startCount      = gbrS.cB.streamCount      = (long)startCount;
    gbrS.theSpecRec         = *specRec;
    gbrS.cB.spec            = gbrS.theSpecRec->spec();
    gbrS.originalMeasure    = lineData.fComputedLen = lineData.fMeasure;

    /* this test is somewhat arbitrary, but we are reaching the
    * outer limits of our unit system, and what we probably
    * have encountered is a horizontally flexible column,
    * which should not be justified!!
    */
    gbrS.allowJustification = ( gbrS.originalMeasure < (LONG_MAX-one_pica) );

    /* a little bullet proofing */
    if ( gbrS.originalMeasure < 0 )
        gbrS.originalMeasure = one_pica * 2;

    if ( lineData.IsHorizontal() )
        gbrS.colShapeRag = (eTSJust)( lineData.fColShapeType & eHorzFlex ? eRagLeft : -1 );
    else
        gbrS.colShapeRag = (eTSJust)( lineData.fColShapeType & eVertFlex ? eRagLeft : -1 );

    gbrS.lastLineLen      = lineData.fLastLineLen;

    gbrS.theLineCount      = lineCount;
    gbrS.cB.breakVal       = eCharBreak;
    gbrS.cB.lineVal        = 0;
    gbrS.cB.breakCount     = 0;
    gbrS.cB.spaceCount     = 0;
    gbrS.cB.trailingSpaces = 0;
    gbrS.cB.wsSpaceCount   = 0;
    gbrS.cB.fillSpCount    = 0;
    gbrS.cB.chCount        = 0;

    gbrS.letterSpaceAdj    = 0;
    gbrS.minRelPosition     = 0;
    gbrS.cB.curBox          = 0;

    initialLead            = lineData.fInitialLead.GetLead();
    initialBaseline         = scCachedStyle::GetCurrentCache().GetBaselineType( );

    gbrS.cB.optGlue         = gbrS.cB.minGlue = gbrS.cB.maxGlue = 0L;
    gbrS.tmpMinGlue         = gbrS.tmpOptGlue = gbrS.tmpMaxGlue = 0L;

    gbrS.cB.specChanged     = false;
    gbrS.firstBox           = gbrS.firstGlue = true;
    gbrS.fNoStartline       = false;
    gbrS.fLastHangable      = 0;
    gbrS.numTargetChars     = 0;
    gbrS.totalTrailingSpace = 0;

    gbrS.desiredMeasure = ::BRKRagControl( gbrS.cB.theChRec, lineData.fOrg.x, lineData.fOrg.y,
        lineData.fMeasure, gbrS.cB.spec, lineCount, linesHyphenat
ed );

    if ( gbrS.desiredMeasure <= 0 ) {
        if ( !gbrS.dcSet )
            count = 0;
        else {
            BRKPlaceLine( lineData.fOrg, lineData.fComputedLen, scCachedStyle::GetCurrentCache().Get
Flowdir() );
            count = gbrS.cB.streamCount - startCount;
            BRKMaxLineVals( lineData, initialLead, initialBaseline, 0 );
        }
    }
    #if _DEBUG
        gbrS.Init();
    #endif
    return eCharBreak;

```

```

    }

    if ( gbrS.cb.theChRec->character == scEndStream ) {
        if ( startCount==0 || (gbrS.cb.theChRec-1)->character!=scHardReturn ) {
            BRKPlaceLine( lineData.fOrg, lineData.fComputedLen, scCachedStyle::GetCurrentCache().Get
Flowdir() );
            count = gbrS.cb.streamCount - startCount;
            BRKMaxLineVals( lineData, initialLead, initialBaseline, 0 );
#ifdef _DEBUG
            gbrS.Init();
#endif

            return eEndStreamBreak;
        }
    }

    switch ( BRKTheLoop() ) {
        case start_of_line:
            break;
        case end_of_stream_reached:
            /* the end of paragraph has been detected */
            /* force justify the line,
             * NOTE: we should probably justify the line
             * if it is close to the desired measure
             * I THINK WE DO
             */
            BRKRepairFinalSpace( );
            lineData.fRagSetting = gbrS.effectiveRag;
            if ( gbrS.colShapeRag != (eTSJust)-1 )
                gbrS.effectiveRag = eRagRight;

            if ( (gbrS.effectiveRag & eRagFlag) == eRagJustified && !gbrS.cb.fillSpCount ) {
                if ( !(gbrS.effectiveRag & (int)eLastLineJust) )
                    lineData.fComputedLen = gbrS.cb.curBox + gbrS.cb.optGlue;
                else {
                    if ( gbrS.allowJustification )
                        BRKJustifyLine( );
                    lineData.fComputedLen = gbrS.desiredMeasure;
                }
                lineData.fOrg.x += gbrS.brkLeftMargin;
                if ( gHiliteSpaces )
                    lineData.fComputedLen += gbrS.totalTrailingSpace;
            }
            else
                BRKPlaceLine( lineData.fOrg, lineData.fComputedLen, scCachedStyle::GetCurrentCache()
GetFlowdir() );
            count = gbrS.cb.streamCount - startCount;
            letterSpace = gbrS.letterSpaceAdj;
            break;
        default:
            case measure_exceeded:
                // the line measure has been exceeded, there are still more
                // characters in the paragraph
                lineData.fRagSetting = gbrS.effectiveRag;
                if ( gbrS.colShapeRag != (eTSJust)-1 )
                    gbrS.effectiveRag = eRagRight;

                if ( (gbrS.effectiveRag & eRagFlag) == eRagJustified && !gbrS.cb.fillSpCount ) {
                    if ( gbrS.allowJustification )
                        BRKJustifyLine( );
                    lineData.fOrg.x += gbrS.brkLeftMargin;
                    lineData.fComputedLen = gbrS.desiredMeasure;
                    if ( gHiliteSpaces )
                        lineData.fComputedLen += gbrS.totalTrailingSpace;
                }
                else
                    BRKPlaceLine( lineData.fOrg, lineData.fComputedLen, scCachedStyle::GetCurrentCache()
.GetFlowdir() );
                count = gbrS.cb.streamCount - startCount;
                letterSpace = gbrS.letterSpaceAdj;
                break;
    }
}

```

```

/* increment the line hyphenated count, used to prevent too many
 * consecutive lines hyphenated
 */
if ( gbrS.lineHyphenated )
    linesHyphenated += 1;
else
    linesHyphenated = 0;

/* set these - the leading may force the line to be replaced & rebroken */
BRKMaxLineVals( lineData, initialLead, initialBaseline, (gbrS.cb.theChRec-1)->character>=scWordS
pace?(gbrS.cb.theChRec-1)->escapement:0 );

for ( ; (long)gbrS.cb.streamCount > (*specRec+1)->offset(); (*specRec)++ )
;

if ( gbrS.foundCharIndent )
    BRKSetCharIndent( chRec, startCount, count, gbrS.letterSpaceAdj );

(gbrS.cb.theChRec-1)->flags.SetLineBreak();
scAssert( count >= 0 );

#ifdef _DEBUG
eBreakType ret = gbrS.cb.breakVal;
gbrS.Init();
return ret;
#else
return gbrS.cb.breakVal;
#endif
}

/*****
static void BRKMaxLineVals( scLINERefData& lineData,
                          MicroPoint initialLead,
                          eFntBaseline baseline,
                          MicroPoint lastChWidth )
{
    scAngle      maxAngle      = lineData.fStartAngle;
    int          i;
    scXRect      dcRect;

    if ( gbrS.dcSet ) {
        /* set the max extents of the drop char before we bash linedata,
         * since the dc has to have the linedata that was in effect at
         * the beginning of the line
         */
        gbrS.dcInfo.dcMinY = MIN( gbrS.dcInfo.dcMinY, lineData.fOrg.y + lineData.fInkExtents.y1 );
        gbrS.dcInfo.dcMaxY = gbrS.dcInfo.dcMaxY - ( gbrS.dcInfo.dcMinY + lineData.fBaselineJump );
        gbrS.dcInfo.dcMaxX -= gbrS.dcInfo.dcMinX;

        dcRect.Set( gbrS.dcInfo.dcMinX, gbrS.dcInfo.dcMinY, gbrS.dcInfo.dcMaxX, gbrS.dcInfo.dcMaxY )
    }

    for ( i = gbrS.cb.lineVal; i--; ) {
        if ( initialLead < gbrS.fMaxLineVals[i].fMaxLead.GetLead() ) {
            if ( lineData.fEndLead.GetLead() < gbrS.fMaxLineVals[i].fMaxLead.GetLead() ) {
                lineData.fEndLead      = gbrS.fMaxLineVals[i].fMaxLead;
                lineData.SetMaxLeadSpec( gbrS.fMaxLineVals[i].fSpecRec->spec() );
            }
        }

        scXRect inkExtents( gbrS.fMaxLineVals[i].fMaxInkExtents );
        inkExtents.Translate( lineData.fOrg );
        lineData.fInkExtents.Union( inkExtents );
    }

    if ( lineData.IsHorizontal() ) {
        lineData.fInkExtents.x1 += gbrS.minRelPosition;
        lineData.fInkExtents.x2 += ( lineData.fComputedLen - lastChWidth );
    }
}

```

```

    }
    else {
        lineData.fInkExtents.y1 += gbrS.minRelPosition;
        lineData.fInkExtents.y2 += ( lineData.fComputedLen - lastChWidth );
    }

#if SCDEBUG > 1
    SCDebugTrace( 4, scString( "BRKMaxLineVals: (%d %d %d %d)\n" ),
        muPoints( lineData.fInkExtents.x1 ), muPoints( lineData.fInkExtents.y1 ),
        muPoints( lineData.fInkExtents.x2 ), muPoints( lineData.fInkExtents.y2 ) );
#endif

    if ( gbrS.dcSet ) {
        /* now union the drop cap extents and the 'line' extents */
        lineData.fInkExtents.Union( dcRect );
    }
}

/*****
static inline Bool BRKExceedVals( MicroPoint adjustableSpace )
{
    return gbrS.cB.minGlue + gbrS.tmpMinGlue > adjustableSpace;
}

/*****
/* this is the routine that effectively does the quality line breaking
 * up to this point we have simply been looking for a condition to
 * have been exceeded, now we may search to find a good break point
 */
static UCS2 BRKLineDecision( MicroPoint )
{
    CandBreak *choice;

    if ( ( gbrS.effectiveRag & eRagFlag ) == eRagJustified )
        choice = BRKHyphenateJust( );
    else
        choice = BRKHyphenateRag( );

    if ( choice->breakCount == 0    &&
        gbrS.cB.breakCount > 1    &&
        choice->streamCount == choice->startCount )
        choice++;

    choice->spaceCount      = (ushort)(choice->spaceCount - choice->trailingSpaces);
    choice->wsSpaceCount    = (ushort)(choice->wsSpaceCount - choice->trailingSpaces);
    gbrS.cB = *choice;

    /* this is a fix for a bug in the character loop,
     * in the loop the spec is incremented before the character is
     * called, if the character forces a line break, the spec at the
     * end of the line ( stored in choice->spec ) is invalid, the following
     * fixes this
     */

    while ( gbrS.cB.theChRec->character &&
        gbrS.cB.lineVal &&
        gbrS.cB.specRec->offset() >= (long)gbrS.cB.streamCount ) {
        gbrS.cB.specRec--;
        gbrS.cB.spec = gbrS.cB.specRec->spec();
        gbrS.cB.lineVal--;
    }

    scCachedStyle::GetCachedStyle( gbrS.cB.spec );

    if ( gbrS.cB.breakVal == eHyphBreak ) {
        gbrS.lineHyphenated = true;
        if ( gbrS.cB.theChRec->flags.IsKernPresent() )
            gbrS.cB.curBox += BRKKernCorrection( gbrS.cB.theChRec );
    }
    else {

```

```

        gbrS.lineHyphenated = false;
        BRKRepairLastSpace( gbrS.cb.theChRec, gbrS.cb.trailingSpaces );
    }

    return gbrS.cb.theChRec->character;
}

/*****
/* we have tripped on a word space, if it is the first word space we
* must do some housekeeping, the first word space test performs two
* operations: 1. it tests to see if we have exceeded the measure and
* 2. it counts actual interword spaces areas, we need to know that
* for microjustification
*/
static inline eBreakEvent bmBRKWordSpace( )
{
    BOOL bFirstGlue = gbrS.firstGlue;

    if ( gbrS.firstGlue ) {
        gbrS.firstBox = true;
        gbrS.firstGlue = false;
        gbrS.cb.wsSpaceCount++;
    }

    gbrS.fNoStartline = false;
    gbrS.fLastHangable = 0;

    gbrS.tmpOptGlue += scCachedStyle::GetCurrentCache().GetOptWord();
    gbrS.tmpMinGlue += scCachedStyle::GetCurrentCache().GetMinWord();
    gbrS.tmpMaxGlue += scCachedStyle::GetCurrentCache().GetMaxWord();

    gbrS.cb.theChRec->escapement = scCachedStyle::GetCurrentCache().GetOptWord();

    gbrS.cb.trailingSpaces++;
    gbrS.cb.spaceCount++;
    gbrS.cb.streamCount++;
    gbrS.cb.theChRec++;

    if (!bFirstGlue)
    {
        MicroPoint adjustableSpace = gbrS.desiredMeasure - gbrS.cb.curBox;
        BRKSetCandBreak( eCharBreak );
        if (BRKExceedVals (adjustableSpace))
        {
            BRKLineDecision (0);
            return BRKExitLoop( );
        }
    }

    return in_line;
}

/*****
/* does much the same as the word space break, except these are characters,
* it also checks for hyphens
*/
static inline void BRKSetFirstBox( )
{
    gbrS.firstGlue      = true;
    gbrS.firstBox       = false;
    gbrS.cb.minGlue     += gbrS.tmpMinGlue;
    gbrS.cb.optGlue     += gbrS.tmpOptGlue;
    gbrS.cb.maxGlue     += gbrS.tmpMaxGlue;
    gbrS.tmpMinGlue     = gbrS.tmpOptGlue = gbrS.tmpMaxGlue = 0;
    gbrS.cb.trailingSpaces = 0;
}

/*****
static inline eBreakEvent bmBRKChar( )

```

```

{
    MicroPoint    adjustableSpace;

    if ( gbrS.cb.theChRec->character >= 256 ) {
        adjustableSpace = gbrS.desiredMeasure - gbrS.cb.curBox;
        BRKSetCandBreak( eCharBreak );

        if ( BRKExceedVals( adjustableSpace ) ) {
            BRKLineDecision( 0 );
            return BRKExitLoop( );
        }
        gbrS.cb.curBox += gbrS.cb.theChRec ->escapement;

        gbrS.cb.chCount++;
        gbrS.cb.streamCount++;
        gbrS.cb.theChRec++;

        return in_line;
    }

    if ( gbrS.firstBox ) {
        adjustableSpace = gbrS.desiredMeasure - gbrS.cb.curBox ;

        /* at the start of every word set a potential break point */
        BRKSetCandBreak( eCharBreak );
        if ( BRKExceedVals( adjustableSpace ) ) {
            BRKLineDecision( 0 );
            return BRKExitLoop( );
        }

        BRKSetFirstBox( );
    }

    gbrS.cb.curBox += gbrS.cb.theChRec ->escapement;

    gbrS.cb.chCount++;
    gbrS.cb.streamCount++;
    gbrS.cb.theChRec++;

    return in_line;
}

//*****
static void getfield( stUnivString& ustr,
                    APPColumn    col,
                    scStream*    stream,
                    uint8        id,
                    TypeSpec&    spec )
{
    clField& field = clField::createField( stream, id );
    field.content( ustr, col, spec );
    field.release();
}

/* ===== */

static inline eBreakEvent bmBRKField()
{
    stUnivString    ustr;
    TypeSpec        spec = gbrS.cb.spec;

    getfield( ustr,
              gbrS.theBreakColH->GetAPPName(),
              gbrS.theBreakColH->GetStream(),
              gbrS.cb.theChRec->flags.GetField(),
              spec );

    if ( gbrS.firstBox ) {
        MicroPoint adjustableSpace = gbrS.desiredMeasure - gbrS.cb.curBox;

        /* at the start of every word set a potential break point */
        BRKSetCandBreak( eCharBreak );
    }
}

```



```

        if ( BRKExceedVals( adj, tableSpace ) ) {
            BRKLineDecision( 0 );
            return BRKExitLoop( );
        }

        BRKSetFirstBox( );
    }

    if ( ustr.len )
        gbrS.cB.theChRec->escapement = UnivStringWidth( ustr, 0, spec );

    gbrS.cB.curBox += gbrS.cB.theChRec->escapement;

    gbrS.cB.chCount++;
    gbrS.cB.streamCount++;
    gbrS.cB.theChRec++;

    return in_line;
}

```

```

/*****

```

```

static inline eBreakEvent BRKLoopBody( )
{
    register ushort theCharacter = gbrS.cB.theChRec->character;
    register eBreakEvent breaktype = in_line;

    /* increment the spec counter if necessary,
     * NOTE - this increments the spec too soon!!!
     * If the next character is the break char then
     * the spec will be inaccurate, there is a fix
     * for this in the line ending decision logic,
     * it is commented as such, the cleaner fix
     * would slow the code down too much
     * (sorry for any confusion WAM)
     */
    if ( gbrS.cB.streamCount >= gbrS.theSpecRec->offset() ) {
        if ( theCharacter != scEndStream ) {
            /* do not advance spec unless we have characters */
            gbrS.cB.spec = BRKUpdateSpec( gbrS.theSpecRec );
            gbrS.theSpecRec++;
        }
    }

    /* dispatch the character to the appropriate routine */
    gbrS.cB.theChRec->flags.ClrVarious();

    if ( ! ( theCharacter & 0xFFC0 ) ) {
        if ( theCharacter == scWordSpace )
            breaktype = bmBRKWordSpace();
        else if ( theCharacter & 0x0030 ) {
            switch ( theCharacter ) {
                case scBreakingHyphen:
                    breaktype = bmBRKHyphen();
                    break;
                default:
                    breaktype = bmBRKChar();
                    break;
            }
        }
        else
            breaktype = (*gbrS.breakMach[theCharacter])();
    }
    else
        breaktype = bmBRKChar();

    return breaktype;
}

```

```

/*****

```

```

static eBreakEvent BRKTheLoop(
{
    eBreakEvent bt;
    while ( ( bt = BRKLoopBody() ) == in_line )
        ;
    return bt;
}

/*****
/* handle breaking of characters that we cannot vector to with thw
* 'breakMach' array of sub-routines
*/

#if 0
static void BRKSpecial( ushort theCharacter )
{
    switch ( theCharacter ) {
        case scFillSpace:
            bmBRKFillSpace( );
            break;
        default:
            bmBRKChar();
            break;
    }
}
#endif

/*****
static short BRKPerformDiscHyphen( CharRecordP theChar,
                                   CharRecordP lastChRec,
                                   Hyphen *hyphens )
{
    UCS2      ch;
    short     charCount      = 1,
            numBreaks       = 0;
    int       j;
    Bool      hitLowerCase   = false;
    Bool      hitUpperCase   = false;

    for ( ; theChar <= lastChRec; theChar++, charCount++ ) {
        ch = theChar->character;

        if ( ch == scBreakingHyphen )
            return 0;

        if ( theChar->flags.IsDiscHyphen() ) {
            if ( charCount >= scCachedStyle::GetCurrentCache().GetPreHyph() ) {
                hyphens[numBreaks].offset = charCount;
                hyphens[numBreaks].rank   = eDiscHyphRank;
                numBreaks++;
            }
        }

        if ( ch < 256 && CTIsAlpha( ch ) ) {
            if ( CTIsLowerCase( ch ) )
                hitLowerCase = true;
            else if ( CTIsUpperCase( ch ) )
                hitUpperCase = true;
        }
        else if ( CTIsSpace( ch ) )
            if ( ch != scFixRelSpace && ch != scFixAbsSpace )
                break;
    }

    if ( ( ! scCachedStyle::GetCurrentCache().GetAcronymHyphs() && ! hitLowerCase ) || ( ! scCachedStyle::GetCurrentCache().GetCaseHyphs() && hitUpperCase ) )
        return 0;

    charCount = (short)(charCount - scCachedStyle::GetCurrentCache().GetPostHyph());
    for ( j = numBreaks - 1; j >= 0 && numBreaks > 0; j-- )
        if ( hyphens[j].offset >= charCount )

```

```

        numBreaks--;
    }
    return numBreaks;
}

/*****

static short BRKPerformHyphenation( CharRecordP firstChRec,
                                   CharRecordP lastChRec,
                                   Hyphen*      hyphens )
{
    CharRecordP theChar;
    int         j;
    UCS2        ch;
    UCS2        hyphWord[64];
    short       hyphArray[64];
    short       hLen          = 0;
    Bool        hitLowerCase  = false;
    Bool        hitUpperCase  = false;
    short       numBreaks     = 0;
    short       charCount     = 1;

    SCmemset( hyphArray, 0, sizeof( short ) * 64 );
    SCmemset( hyphWord, 0, sizeof( UCS2 ) * 64 );

    for ( theChar = firstChRec; theChar <= lastChRec; theChar++ ) {
        ch = theChar->character;
        if ( ch < 256 && CTIsAlpha( ch ) )
            break;
    }

    for ( ; theChar <= lastChRec; theChar++, charCount++ ) {
        ch = theChar->character;

        if ( ch == scBreakingHyphen ) {
            return 0;
        }
        else if ( theChar->flags.IsDiscHyphen() ) {
            return BRKPerformDiscHyphen( firstChRec, lastChRec, hyphens );
        }
        else if ( ch < 256 && CTIsAlpha( ch ) ) {
            if ( hLen < 63 ) {
                if ( CTIsLowerCase( ch ) )
                    hitLowerCase = true;
                else if ( CTIsUpperCase( ch ) )
                    hitUpperCase = true;
                hyphWord[ hLen++ ] = CTToLower( ch );
            }
        }
        else if ( ch != scFixRelSpace && ch != scFixAbsSpace ) { /* hit delimiter */
            break;
        }
    }

    if ( hLen < scCachedStyle::GetCurrentCache().GetMaxWordHyph()
        || ( ! scCachedStyle::GetCurrentCache().GetAcronymHyphs() && ! hitLowerCase )
        || ( ! scCachedStyle::GetCurrentCache().GetCaseHyphs() && hitUpperCase ) )
        return 0;

    if ( HYFWord( hyphWord, hyphArray ) ) {
        theChar = firstChRec;
        for ( j = 0, charCount = 1; j < hLen; charCount++, theChar++ ) {
            ch = CTToLower( theChar->character );
            if ( ch == hyphWord[j] ) {
                if ( hyphArray[j] && j >= ( scCachedStyle::GetCurrentCache().GetPreHyph() - 1 ) && j
                    < ( hLen - scCachedStyle::GetCurrentCache().GetPostHyph() ) ) {
                    switch( hyphArray[j] & 0x3f ) {
                        case 1:
                        case 2:
                            hyphens[numBreaks].rank = eBestHyphRank;
                            break;
                        case 3:

```

```

        hyphens[numBreaks].rank = eGoodHyphRank;
        break;
    default:
        hyphens[numBreaks].rank = eBadHyphRank;
        break;
    }
    hyphens[numBreaks].offset = charCount;
    numBreaks++;
}
j++;
}
}

return numBreaks;
}

/*****
/* This is called only in the case where even the first word (or the
/* first legal portion of it before a hyphen) will not fit on the line.
/* Therefore, we add a hyphen after every character to force a break.
*/

#ifdef scForceBreakFirstWord

static void BRKForceHyphens( CandBreak *startBreak,
                             CandBreak *endBreak )
{
    CandBreak    savedEndBreak;

    savedEndBreak = *endBreak;

    /* We are resetting the break machine to the start of the word. */
    /* The values from this point on in the machine will be over-    */
    /* written.                                                         */

    gbrS.tmpMinGlue = (startBreak+1)->minGlue - startBreak->minGlue;
    gbrS.tmpOptGlue = (startBreak+1)->optGlue - startBreak->optGlue;
    gbrS.tmpMaxGlue = (startBreak+1)->maxGlue - startBreak->maxGlue;
    gbrS.cB         = *startBreak;
    gbrS.firstBox   = true;      /* signal start of word */

    while ( gbrS.theSpecRec->offset() > gbrS.cB.streamCount ) // reset the spec to the start of the
    word
        gbrS.theSpecRec--;

    while ( true ) {
        BRKLoopBody( );

        if ( gbrS.cB.chCount == savedEndBreak.chCount )
            break;

        BRKSetCandBreak( eHyphBreak );
        if ( scCachedStyle::GetCurrentCache().GetHyphLanguage() != Japanese )
            (gbrS.cB.theChRec - 1)->flags.SetAutoHyphen( eGoodHyphRank );

        if ( ( gbrS.effectiveRag & eRagFlag ) == eRagJustified ) {
            if ( gbrS.cB.curBox + gbrS.cB.minGlue > gbrS.desiredMeasure )
                break;
        }
        else {
            if ( gbrS.cB.curBox + gbrS.cB.optGlue > gbrS.desiredMeasure )
                break;
        }
    }

    savedEndBreak.breakCount = gbrS.cB.breakCount;
    gbrS.candBreak[gbrS.cB.breakCount] = savedEndBreak;

    if ( gbrS.cB.breakCount >= (MAXBREAKVALS-1) )
        ShuffleBreakCandidates();
    else
        gbrS.cB.breakCount++;
}

```

```

}

#endif /* forceBreakFirstWord */

/*****

#define NotHyphBreak( c )      ( ((c)!=eHyphBreak) && ((c)!=eHardHyphBreak) )
#define useBadHyphens         false

static void BRKAddHyphens( CandBreak *startBreak,
                          CandBreak *endBreak )
{
    CandBreak    savedEndBreak;
    CharRecordP  startChRec;
    CharRecordP  stopChRec;
    Hyphen       hyphens[64];
    short        i,
                offset,
                prevOffset    = 0,
                numHyphens;

    if ( startBreak == endBreak || endBreak->curBox + endBreak->optGlue <= gbrS.desiredMeasure )
        return;

    if ( ! gbrS.allowHyphens || ! scCachedStyle::GetCurrentCache().GetHyphenate() ) {
#ifdef scForceBreakFirstWord
        if ( startBreak == &gbrS.candBreak[0] && startBreak->streamCount == startBreak->startCount )
            BRKForceHyphens( startBreak, endBreak );
#endif
        return;
    }

    savedEndBreak    = *endBreak;
    startChRec       = startBreak->theChRec;
    stopChRec        = endBreak->theChRec;

    numHyphens = BRKPerformHyphenation( startChRec, stopChRec, hyphens );
    if ( numHyphens == 0 ) {
#ifdef scForceBreakFirstWord
        if ( startBreak == &gbrS.candBreak[0] && startBreak->streamCount == startBreak->startCount )
            BRKForceHyphens( startBreak, endBreak );
#endif
        return;
    }

    /* We are resetting the break machine to the start of the word. */
    /* The values from this point on in the machine will be over- */
    /* written. The end of word break, since it is past the measure, */
    /* will never be restored, and the final break will now be the */
    /* last hyphen break. */
    /* Use endBreak to return the last hyphen breakpoint we find. */

    gbrS.tmpMinGlue = (startBreak+1)->minGlue - startBreak->minGlue;
    gbrS.tmpOptGlue = (startBreak+1)->optGlue - startBreak->optGlue;
    gbrS.tmpMaxGlue = (startBreak+1)->maxGlue - startBreak->maxGlue;
    gbrS.cB         = *startBreak;
    gbrS.firstBox   = true;      /* signal start of word */

    for ( i = 0; i < numHyphens; i++ ) {
        if ( ! useBadHyphens && hyphens[i].rank == eBadHyphRank )
            continue;

        offset = hyphens[i].offset;
        for ( ; prevOffset < offset; prevOffset++ )
            BRKLoopBody( );

        BRKSetCandBreak( eHyphBreak );

        if ( !(gbrS.cB.theChRec-1)->flags.IsDiscHyphen() )

```

```

        (gbrS.cb.theChRec-1)->f.SetAutoHyphen( hyphens[i].rank )

        if ( ( gbrS.effectiveRag & eRagFlag ) == eRagJustified ) {
            if ( gbrS.cb.curBox + gbrS.cb.minGlue > gbrS.desiredMeasure )
                break;
        }
        else {
            if ( gbrS.cb.curBox + gbrS.cb.optGlue > gbrS.desiredMeasure )
                break;
        }
    }

#ifdef scForceBreakFirstWord
    /* if the first hyphen exceeded the measure, we may want to force a break. */
    if ( i == 0 && startBreak == &gbrS.candBreak[ 0 ]
        && startBreak->streamCount == startBreak->startCount ) {
        BRKForceHyphens( startBreak, endBreak );
        return;
    }
#endif

    savedEndBreak.breakCount = gbrS.cb.breakCount;
    gbrS.candBreak[gbrS.cb.breakCount] = savedEndBreak;

    if ( gbrS.cb.breakCount >= (MAXBREAKVALS-1) )
        ShuffleBreakCandidates();
    else
        gbrS.cb.breakCount++;
}

/*****
static CandBreak *BRKHyphenateRag( )
{
    CandBreak *theBreak,
    *choice = NULL;
    CandBreak *startWord = NULL,
    *endWord;
    long bCount = gbrS.cb.breakCount;
    MicroPoint diff,
    bestDiff = LONG_MAX,
    lineSpace;
    Bool lineDiff = true; /* irrelevant unless otherwise set */

    /* If there is a hyphenation zone, first try to find a non-hyphen break.
    * If that fails, find the start and end of the word straddling the
    * line break, call hyphenation routine, and call line decision to
    * find the best break.
    */

    if ( gbrS.hyphenationZone ) {
        theBreak = gbrS.candBreak + gbrS.cb.breakCount - 1;

        for ( ; bCount-- > 0; theBreak-- ) {

            lineSpace = theBreak->curBox + theBreak->optGlue;
            if ( scCachedStyle::GetCurrentCache().GetDiffRagZone() )
                lineDiff = ABS( gbrS.lastLineLen - lineSpace ) > scCachedStyle::GetCurrentCache(
).GetDiffRagZone();

            /* look for a non-hyphen break and a line with a sufficient diff zone */
            if ( NotHyphBreak( theBreak->breakVal )
                && gbrS.desiredMeasure >= lineSpace
                && gbrS.desiredMeasure <= lineSpace + gbrS.hyphenationZone
                && lineDiff )
            {
                choice = theBreak;
                break;
            }
        }
    }
}

```

```

    if ( choice )
        return choice;

/* First, get closest word break greater than desired measure */

bCount      = gbrS.cB.breakCount;
endWord      = theBreak = gbrS.candBreak + gbrS.cB.breakCount - 1;
for ( ; bCount-- > 0; theBreak-- ) {

    diff      = theBreak->curBox + theBreak->optGlue - gbrS.desiredMeasure;
    if ( diff <= 0 )
        break;

    if ( NotHyphBreak( theBreak->breakVal ) && diff < bestDiff ) {
        bestDiff  = diff;
        endWord    = theBreak;
    }
}

/* Next, get closest word break less than desired measure */

startWord    = endWord;
bCount++;
/* reset it to what it should be */
for ( ; bCount-- > 0; theBreak-- ) {
    if ( NotHyphBreak( theBreak->breakVal ) ) {
        startWord = theBreak;
        break;
    }
}

BRKAddHyphens( startWord, endWord );
return BRKLineDecisionRag( );
}

/*****
static CandBreak *BRKHyphenateJust( )
{
    CandBreak      *theBreak;
    CandBreak      *choice      = NULL,
                  *startWord    = NULL,
                  *endWord;
    long           bCount      = gbrS.cB.breakCount;
    long           endCount;
    MicroPoint     adjustableSpace,
                  diff,
                  bestDiff      = LONG_MAX;
    endWord        = theBreak = gbrS.candBreak + bCount - 1;
    for ( ; bCount-- > 0; theBreak-- ) {

        /* amount of space we have to play with */
        adjustableSpace = gbrS.desiredMeasure + theBreak->fHangable - theBreak->curBox;

        diff = adjustableSpace - theBreak->optGlue;
        if ( diff <= 0 && NotHyphBreak( theBreak->breakVal ) ) {
            endWord    = theBreak;
            endCount    = bCount + 1;
        }

        diff = ABS( diff );

        if ( NotHyphBreak( theBreak->breakVal ) && diff < bestDiff ) {
            if ( adjustableSpace <= theBreak->maxGlue && adjustableSpace >= theBreak->minGlue ) {
                choice    = theBreak;
                bestDiff  = diff;
            }
        }
        if ( diff > bestDiff )
            break;
    }
}

if ( choice )

```

```

    return choice;

    theBreak    = startWord = endWord;
    bCount      = endCount;      /* restore its last value */
    for ( ; bCount-- > 0; theBreak-- ) {
        if ( NotHyphBreak( theBreak->breakVal )
            && theBreak->curBox + theBreak->optGlue < gbrS.desiredMeasure + theBreak->fHangabl
e ) {
            startWord    = theBreak;
            break;
        }
    }

    BRKAddHyphens( startWord, endWord );
    return BRKLineDecisionJust( );
}

/*****/

static CandBreak *BRKLineDecisionRag( )
{
    CandBreak    *theBreak,
                 *choice      = NULL;
    long         bCount      = gbrS.cB.breakCount;
    MicroPoint   lineSpace;

    /* there is a very strange bug here in that the char plus hyphen may be
     * chosen instead of the entire word because the hyphen may be wider than
     * the trailing letter(s)
     */
    /* Note: We only get here if hyphenationZone is off or we failed to find
     * a good non-hyphen break.
     */
    if ( scCachedStyle::GetCurrentCache().GetDiffRagZone() ) {
        theBreak    = gbrS.candBreak + gbrS.cB.breakCount - 1;

        for ( ; bCount-- > 0; theBreak-- ) {
            lineSpace    = theBreak->curBox + theBreak->optGlue;
            /* this reflects space changed by hyphenation spelling changes */

            if ( gbrS.desiredMeasure >= lineSpace && ABS( gbrS.lastLineLen - lineSpace ) > scCached
Style::GetCurrentCache().GetDiffRagZone() ) {
                choice    = theBreak;
                break;
            }
        }
    }

    /* resort to worst case if necessary */
    /* if !GetDiffRagZone() && !hyphenationZone, we will fall through to here */

    if ( choice == NULL ) {
        bCount      = gbrS.cB.breakCount;
        choice      = theBreak    = gbrS.candBreak + gbrS.cB.breakCount - 1;

        for ( ; bCount-- > 0; theBreak-- ) {
            lineSpace    = theBreak->curBox + theBreak->optGlue;
            /* this reflects space changed by hyphenation spelling changes */

            if ( gbrS.desiredMeasure + theBreak->fHangable >= lineSpace ) {
                choice    = theBreak;
                break;
            }
        }
    }

    return choice;
}

```



```

.....

static CandBreak *BRKLineDecisionJust( )
{
    CandBreak    *theBreak,
                 *choice;
    long    bCount    = gbrS.cB.breakCount;
    MicroPoint    lineSpace,
                 diff,
                 bestDiff    = LONG_MAX;

    choice    = theBreak    = gbrS.candBreak + bCount - 1;

    for ( ; bCount-- > 0; theBreak-- ) {
        lineSpace    = theBreak->curBox + theBreak->minGlue;
        /* this reflects space changed by hyphenation spelling changes */

        diff    = lineSpace - gbrS.desiredMeasure;
        if ( diff < theBreak->fHangable && ABS( diff ) < bestDiff ) {
            choice    = theBreak;
            bestDiff    = ABS( diff );
        }
        if ( diff > bestDiff )
            break;
    }

    return choice;
}

.....
/* if we break on a hyphenation point and the character is kerned with
 * the next character we have an incorrect line length, because of the
 * kern built into the characters escapement, here we get that correction
 */

static MicroPoint BRKKernCorrection( CharRecordP aChRec )
{
    return ( scCachedStyle::GetCurrentCache().GetEscapement( aChRec->character ) - aChRec->escapemen
t );
}

.....
/* For non-justified lines this places the line */

static void BRKPlaceLine( scMuPoint&                lineOrigin,
                         MicroPoint&                measure,
                         const scFlowDir& fd )
{
    MicroPoint    actualMeasure,
                 translation;

    actualMeasure = gbrS.cB.curBox + gbrS.cB.optGlue;
    if ( gbrS.cB.fillSpCount ) {
        MicroPoint    fill;
        long    count;
        CharRecordP    tmpChRec;

        fill = gbrS.desiredMeasure - actualMeasure;
        fill = scRoundMP( (REAL)fill / gbrS.cB.fillSpCount );

        actualMeasure = gbrS.desiredMeasure;
        for ( tmpChRec = gbrS.cB.theChRec, count = gbrS.cB.fillSpCount;
              tmpChRec >= gbrS.gStartRec && count; tmpChRec-- ) {
            if ( tmpChRec->character == scFillSpace )
                tmpChRec->escapement = (GlyphSize)fill;
        }
    }

    switch ( gbrS.effectiveRag & eRagFlag ) {
    default:
    case eRagRight:
        translation = gbrS.brkLeftMargin;
        break;
    }
}

```

```

case eRagLeft:
    if ( gbrS.effectiveRag & (int)eHangPuncRight )
        gbrS.desiredMeasure += BRKHangPuncRightAdjust( );
    translation = gbrS.desiredMeasure - actualMeasure + gbrS.brkLeftMargin;

    if ( !gbrS.lineHyphenated ) {
        /* this accounts for any track kerning - no need to worry about with
        * hyphenation because, we want trackkerning between the hyphen
        * and the last character - and the hyphen escapement does not
        * include any track kerning
        */
        translation += scCachedStyle::GetCurrentCache().GetOptLSP();
    }
    break;
case eRagCentered:
    if ( gbrS.lineHyphenated )
        translation = scRoundMP( (REAL)( gbrS.desiredMeasure - actualMeasure ) / 2 ) + gbrS.brkLeftMargin;
    else
        translation = scRoundMP( (REAL)( gbrS.desiredMeasure - actualMeasure + scCachedStyle::GetCurrentCache().GetOptLSP() ) / 2 ) + gbrS.brkLeftMargin;
    break;
}

if ( fd.IsHorizontal() )
    lineOrigin.Translate( translation, 0 );
else
    lineOrigin.Translate( 0, translation );

if ( gHiliteSpaces )
    actualMeasure += gbrS.totalTrailingSpace;
measure = actualMeasure;
}

/* *****
/* handle adjustment for hanging punctuation on the right */
/* *****

static MicroPoint BRKHangPuncRightAdjust( )
{
    CharRecordP lastCharOnLine;

    if ( gbrS.lineHyphenated )
        return scCachedStyle::GetCurrentCache().GetRightHangValue( scCachedStyle::GetCurrentCache().GetHyphChar() );

    lastCharOnLine = BRKLastCharOnLine( gbrS.cb.theChRec - 1 );
    if ( CTIsPunc( lastCharOnLine->character ) )
        return scCachedStyle::GetCurrentCache().GetRightHangValue( lastCharOnLine->character );
    else
        return 0L;
}

/* *****

#define COMP_LETTERSPACES( charCount, spaceCount, chRec ) \
    ((long)( charCount-spaceCount-1 + ((chRec-1)->flags.IsHyphPresent() ? 1:0) ))

static void BRKJustifyLine( )
{
    GlyphSize    glueSpace,
                 adjustableSpace;
    long         lspSpaces;

    if ( gbrS.effectiveRag & (int)eHangPuncRight )
        gbrS.desiredMeasure += BRKHangPuncRightAdjust( );

    glueSpace = (GlyphSize)(gbrS.desiredMeasure - gbrS.cb.curBox);

    if ( glueSpace < 0 ) {
        gbrS.desiredMeasure += gbrS.cb.fHangable;
        glueSpace = (GlyphSize)(gbrS.desiredMeasure - gbrS.cb.curBox);
    }
}

```

```

if ( !gbrS.lineHyphenated )
    glueSpace += scCachedStyle::GetCurrentCache().GetOptLSP();

if ( gbrS.cb.spaceCount ) {
    gbrS.justSpace = scRoundGS( (REAL)glueSpace / gbrS.cb.spaceCount );

    if ( gbrS.justSpace < 0 ) {
        lspSpaces = COMP_LETTERSPACES( gbrS.cb.chCount, 0, gbrS.cb.theChRec);
        if ( lspSpaces )
            gbrS.letterSpaceAdj = scRoundGS( (REAL)glueSpace / lspSpaces );
        else
            gbrS.letterSpaceAdj = glueSpace;
        gbrS.justSpace = 0;
    }
    else if ( gbrS.justSpace > scCachedStyle::GetCurrentCache().GetMaxWord() ) {
        lspSpaces = COMP_LETTERSPACES( gbrS.cb.chCount, gbrS.cb.spaceCount, gbrS.cb.theCh
Rec);
        adjustableSpace = MPtoGS( glueSpace - gbrS.cb.maxGlue );
        if ( lspSpaces )
            gbrS.letterSpaceAdj = scRoundGS( (REAL)adjustableSpace / lspSpaces );
        else
            gbrS.letterSpaceAdj = adjustableSpace;
        if ( gbrS.letterSpaceAdj < scCachedStyle::GetCurrentCache().GetMinLSP() ) {
            gbrS.letterSpaceAdj = scCachedStyle::GetCurrentCache().GetMinLSP();
            adjustableSpace = MPtoGS( glueSpace - scRoundMP( (REAL)gbrS.letterSpaceAdj * lspSpac
es ) );
            gbrS.justSpace = scRoundGS( (REAL)adjustableSpace / gbrS.cb.spaceCount );
        }
        else if ( gbrS.letterSpaceAdj > scCachedStyle::GetCurrentCache().GetMaxLSP() ) {
            gbrS.letterSpaceAdj = scCachedStyle::GetCurrentCache().GetMaxLSP();
            adjustableSpace = MPtoGS( glueSpace - scRoundMP( (REAL)gbrS.letterSpaceAdj * lspSpac
es ) );
            gbrS.justSpace = scRoundGS( (REAL)adjustableSpace / gbrS.cb.spaceCount );
        }
        else
            gbrS.justSpace = scCachedStyle::GetCurrentCache().GetMaxWord();
    }
    else if ( gbrS.justSpace < scCachedStyle::GetCurrentCache().GetMinWord() ) {
        lspSpaces = COMP_LETTERSPACES( gbrS.cb.chCount, gbrS.cb.spaceCount, gbrS.cb.theChr
ec);
        adjustableSpace = MPtoGS( glueSpace - gbrS.cb.minGlue );
        if ( lspSpaces )
            gbrS.letterSpaceAdj = scRoundGS( (REAL)adjustableSpace / lspSpaces );
        else
            gbrS.letterSpaceAdj = adjustableSpace;
        if ( gbrS.letterSpaceAdj < scCachedStyle::GetCurrentCache().GetMinLSP() ) {
            gbrS.letterSpaceAdj = scCachedStyle::GetCurrentCache().GetMinLSP();
            adjustableSpace = MPtoGS( glueSpace - scRoundMP( (REAL)gbrS.letterSpaceAdj * lspSpac
es ) );
            gbrS.justSpace = scRoundGS( (REAL)adjustableSpace / gbrS.cb.spaceCount );
        }
        else if ( gbrS.letterSpaceAdj > scCachedStyle::GetCurrentCache().GetMaxLSP() ) {
            gbrS.letterSpaceAdj = scCachedStyle::GetCurrentCache().GetMaxLSP();
            adjustableSpace = MPtoGS( glueSpace - scRoundMP( (REAL)gbrS.letterSpaceAdj / lspSpac
es ) );
            gbrS.justSpace = scRoundGS( (REAL)adjustableSpace / gbrS.cb.spaceCount );
        }
        else
            gbrS.justSpace = scCachedStyle::GetCurrentCache().GetMinWord();
    }
    BRKAdjustWordSpace( gbrS.cb.theChRec,
        gbrS.justSpace, gbrS.cb.spaceCount,
        gbrS.cb.trailingSpaces );
}
else {
    lspSpaces = COMP_LETTERSPACES( gbrS.cb.chCount, 0, gbrS.cb.theChRec);
    if ( lspSpaces )
        gbrS.letterSpaceAdj = scRoundGS( (REAL)glueSpace / lspSpaces );
    else

```

```
gbrS.letterSpaceAdj = gbrS.space;
```

```
#ifdef LimitLetterSpace
```

```
/* should we constrain this to min/max letterspace */
```

```
gbrS.letterSpaceAdj = MIN( gbrS.letterSpaceAdj, scCachedStyle::GetCurrentCache().GetMaxLSP()
```

```
);
```

```
gbrS.letterSpaceAdj = MAX( gbrS.letterSpaceAdj, scCachedStyle::GetCurrentCache().GetMinLSP()
```

```
);
```

```
#endif
```

```
}
```

```
}
```

```
/******
```

```
static eBreakEvent bmBRKFixSpace( )
```

```
{
```

```
MicroPoint adjustableSpace;
```

```
if ( gbrS.firstBox ) {
```

```
adjustableSpace = gbrS.desiredMeasure - gbrS.cB.curBox;
```

```
/* at the start of every word set a potential break point */
```

```
BRKSetCandBreak( eCharBreak );
```

```
if ( BRKExceedVals( adjustableSpace ) ) {
```

```
BRKLineDecision( 0 );
```

```
return BRKExitLoop( );
```

```
}
```

```
BRKSetFirstBox();
```

```
gbrS.firstGlue = true;
gbrS.firstBox = false;
gbrS.cB.minGlue += gbrS.tmpMinGlue;
gbrS.cB.optGlue += gbrS.tmpOptGlue;
gbrS.cB.maxGlue += gbrS.tmpMaxGlue;
gbrS.tmpMinGlue = gbrS.tmpOptGlue = gbrS.tmpMaxGlue = 0;
gbrS.cB.trailingSpaces = 0;
```

```
gbrS.fNoStartline = false;
```

```
gbrS.fLastHangable = 0;
```

```
gbrS.cB.curBox += gbrS.cB.theChRec->escapement;
```

```
gbrS.cB.theChRec++;
```

```
gbrS.cB.streamCount++;
```

```
return in_line;
```

```
}
```

```
/******
```

```
static eBreakEvent bmBRKRelSpace( )
```

```
{
```

```
MicroPoint adjustableSpace;
```

```
if ( gbrS.firstBox ) {
```

```
adjustableSpace = gbrS.desiredMeasure - gbrS.cB.curBox;
```

```
/* at the start of every word set a potential break point */
```

```
BRKSetCandBreak( eCharBreak );
```

```
if ( BRKExceedVals( adjustableSpace ) ) {
```

```
BRKLineDecision( 0 );
```

```
return BRKExitLoop( );
```

```
}
```

```
BRKSetFirstBox();
```

```
gbrS.firstGlue = true;
gbrS.firstBox = false;
gbrS.cB.minGlue += gbrS.tmpMinGlue;
gbrS.cB.optGlue += gbrS.tmpOptGlue;
gbrS.cB.maxGlue += gbrS.tmpMaxGlue;
gbrS.tmpMinGlue = gbrS.tmpOptGlue = gbrS.tmpMaxGlue = 0;
gbrS.cB.trailingSpaces = 0;
```

```
}
```

```

gbrS.fNoStartline = false;
gbrS.fLastHangable = 0;

gbrS.cb.curBox += SCRLUCompMP( scCachedStyle::GetCurrentCache().GetGlyphWidth(), (RLU) gbrS.cb.theChRec->escapement );
gbrS.minRelPosition = MIN( gbrS.cb.curBox, gbrS.minRelPosition );
gbrS.cb.theChRec++;
gbrS.cb.streamCount++;

return in_line;
}

/*****/

static eBreakEvent bmBRKHardReturn( )
{
    gbrS.cb.theChRec->escapement = 0;
    gbrS.cb.theChRec++;
    gbrS.cb.streamCount++;

    BRKSetCandBreak( eSpaceBreak );
    BRKLineDecision( 0 );
    return BRKExitLoop( );
}

/*****/

static eBreakEvent bmBRKQuad( )
{
    gbrS.cb.theChRec->escapement = 0;
    gbrS.cb.theChRec++;
    gbrS.cb.streamCount++;
    if ( gbrS.cb.theChRec->character == scEndStream )
        BRKSetCandBreak( eEndStreamBreak );
    else
        BRKSetCandBreak( eSpaceBreak );
    BRKLineDecision( 0 );
    return BRKExitLoop( );
}

/*****/

static eBreakEvent bmBRKVertTab( )
{
    gbrS.cb.theChRec->escapement = 0;
    gbrS.cb.theChRec++;
    gbrS.cb.streamCount++;
    BRKSetCandBreak( eColumnBreak );

    BRKLineDecision( 0 );
    return BRKExitLoop( );
}

/*****/

static Bool TabBreakChar( UCS2 theCh,
                          UCS2 breakCh )
{
    if ( breakCh == theCh )
        return true;
    else {
        switch ( theCh ) {
            default:
                return false;
            case scEndStream:
                /*
                case wordSpace:*/
                case scTabSpace:
                case scFillSpace:
                /* vetical breaks */
                case scVertTab:
                /* horizontal breaks */

```

```

        case scHardReturn:
        case scQuadCenter:
        case scQuadLeft:
        case scQuadRight:
        case scQuadJustify:
            return true;
    }
}

```

```

/*****

```

```

static MicroPoint BRKNextTokenWidth( CharRecordP chRec,
                                     UCS2 breakCh )
{
    MicroPoint tokenWidth = 0;
    MicroPoint charWidth;
    UCS2 theCh;
    long tStreamCount = gbrS.cb.streamCount;
    scSpecRecord * curSpecRec = gbrS.theSpecRec;

    for ( theCh = chRec->character;
          !TabBreakChar(theCh,breakCh);
          chRec++,theCh = chRec->character ) {

        if ( (long)tStreamCount >= gbrS.theSpecRec->offset() ) {
            gbrS.cb.spec
                = BRKUpdateSpec( gbrS.theSpecRec );
            gbrS.theSpecRec++;
        }
        switch ( theCh ) {
            default:
                charWidth = chRec->escapement;
                break;
            case scWordSpace:
                charWidth = scCachedStyle::GetCurrentCache().GetOptWord();
                break;
            case scFixRelSpace:
                charWidth = SCRLUCompGS( scCachedStyle::GetCurrentCache().GetSetSize(),(RLU)chRe
c->Escapement );
                break;
        }

        tokenWidth += charWidth;
    }

    if ( curSpecRec != gbrS.theSpecRec ) {
        gbrS.theSpecRec = curSpecRec;
        gbrS.cb.spec
            = BRKUpdateSpec( gbrS.theSpecRec );
    }

    return tokenWidth;
}

```

```

/*****

```

```

static void BRKSetCharIndent(
    CharRecordP chRec, /* the character array */
    long startCount, /* count into char array that starts line */
    long count, /* count into char array of end of line */
    MicroPoint letterSpace )
{
    MicroPoint indent;

    for ( indent = 0, chRec += startCount; count-->0; chRec++ ) {
        switch ( chRec->character ) {
            default:
                if ( LETTERSPACE( chRec ) )
                    indent += (chRec->escapement + letterSpace);
                else

```

```

        indent += chRec->escapement;
        break;
    case scFixRelSpace:
        indent += SCRLUCompMP( scCachedStyle::GetCurrentCache().GetGlyphWidth(), (RLU)chRec-
>escapement );
        break;
    }
}
gbrS.foundCharIndent = false;
}

/*****/

static eBreakEvent bmBRKTab( )
{
    scTabInfo    tabInfo;
    MicroPoint    currentPosition,
                nextTokenWidth = 0,
                alignTokenWidth = 0;
    CharRecordP    tabChRec      = gbrS.cb.theChRec;

    BRKSetCandBreak( eCharBreak );

    tabChRec->escapement = 0;
    currentPosition = gbrS.cb.curBox +
                    gbrS.cb.optGlue + gbrS.tmpOptGlue +
                    gbrS.brkLeftMargin + gbrS.theLineOrg;

    TSTabInfo( gbrS.pspec_,
                gbrS.cb.spec,
                tabInfo,
                currentPosition,
                0,
                gbrS.theLineCount );

    switch ( tabInfo.tabAlign ) {
        default:
            case eTBLeftAlign:
                break;
            case eTBRightAlign:
                alignTokenWidth = nextTokenWidth = BRKNextTokenWidth( gbrS.cb.theChRec+1, '\0' );
                break;
            case eTBDecimalAlign:
                alignTokenWidth = BRKNextTokenWidth( gbrS.cb.theChRec + 1, scCachedStyle::GetCurrentCach
e().GetDecimalChar() );

                nextTokenWidth = BRKNextTokenWidth( gbrS.cb.theChRec+1, '\0' );
                break;
            case eTBCenterAlign:
                nextTokenWidth = BRKNextTokenWidth( gbrS.cb.theChRec + 1, '\0' );
                alignTokenWidth = scRoundMP( (REAL)nextTokenWidth / 2 );
                break;
    }

    tabChRec->escapement = (GlyphSize)(tabInfo.xPosition - currentPosition - alignTokenWidth);

    if ( tabChRec->escapement < 0 ) {
        alignTokenWidth += tabChRec->escapement;          /* wam added 7/22 */
        tabChRec->escapement = 0;
    }

    if ( gbrS.desiredMeasure < currentPosition + tabChRec->escapement + nextTokenWidth ) {
        if ( gbrS.cb.curBox + gbrS.cb.optGlue + gbrS.tmpOptGlue > 0 ) {
            BRKLineDecision( 0 );
            return BRKExitLoop( );
        }
    }

    gbrS.cb.curBox = tabInfo.xPosition - ( gbrS.brkLeftMargin + gbrS.theLineOrg ) - alignTokenWid
th;

    gbrS.cb.theChRec++;
    gbrS.cb.streamCount++;

```

```

gbrS.firstGlue      = true;
gbrS.firstBox       = true;      /* sil to true on 6/12/92 */
gbrS.fNoStartline   = false;
gbrS.fLastHangable  = 0;

gbrS.cB.minGlue     = gbrS.cB.optGlue = gbrS.cB.maxGlue = 0;
gbrS.tmpMinGlue     = gbrS.tmpOptGlue = gbrS.tmpMaxGlue = 0;
gbrS.cB.trailingSpaces = 0;
gbrS.cB.wsSpaceCount = 0;
gbrS.cB.spaceCount   = 0;

    // define noLeftAlignTabbedLines and this
    // will allow tabbed lines to be none left aligned,
    // the manager of the spec system had better
    // guarantee that the values are reasonable
#ifdef noLeftAlignTabbedLines
    gbrS.cB.fillSpCount++;
#endif

    return in_line;
}

/*****/

static eBreakEvent bmBRKRule( void )
{
    if ( gbrS.cB.curBox ) {
        BRKSetCandBreak( eCharBreak );
        BRKLineDecision( 0 );
        return BRKExitLoop( );
    }

    CharRecordP chRec = gbrS.cB.theChRec;
    chRec->escapement = gbrS.desiredMeasure;
    gbrS.cB.curBox += gbrS.cB.theChRec->escapement;
    gbrS.cB.streamCount++;
    gbrS.cB.theChRec++;

    if ( gbrS.cB.theChRec->character ) {
        BRKSetCandBreak( eSpaceBreak );
        BRKLineDecision( 0 );
        return BRKExitLoop( );
    }

    return in_line;
}

/*****/

static eBreakEvent bmBRKFillSpace( )
{
    MicroPoint adjustableSpace;

    if ( gbrS.firstBox ) {
        adjustableSpace = gbrS.desiredMeasure - gbrS.cB.curBox;

        BRKSetCandBreak( eCharBreak );
        if ( gbrS.cB.minGlue > adjustableSpace ) {
            BRKLineDecision( 0 );
            return BRKExitLoop( );
        }
    }
    BRKSetFirstBox();
    gbrS.firstGlue      = true;
    gbrS.firstBox       = false;
    gbrS.cB.minGlue     += gbrS.tmpMinGlue;
    gbrS.cB.optGlue     += gbrS.tmpOptGlue;
    gbrS.cB.maxGlue     += gbrS.tmpMaxGlue;
    gbrS.tmpMinGlue     = gbrS.tmpOptGlue = gbrS.tmpMaxGlue = 0;
    gbrS.cB.trailingSpaces = 0;
}

```



```

    gbrS.fNoStartline = false;
    gbrS.fLastHangable = 0;

    gbrS.cB.streamCount++;
    gbrS.cB.fillSpCount++;
    gbrS.cB.theChRec->escapement = 0;
    gbrS.cB.theChRec++;

    return in_line;
}

/*****/

static eBreakEvent bmBRKHyphen( )
{
    MicroPoint adjustableSpace;

    if ( gbrS.firstBox ) {
        adjustableSpace = gbrS.desiredMeasure - gbrS.cB.curBox;

        BRKSetCandBreak( eCharBreak );
        if ( gbrS.cB.minGlue > adjustableSpace ) {
            BRKLineDecision( 0 );
            return BRKExitLoop( );
        }
        BRKSetFirstBox();
        // gbrS.firstGlue      = true;
        // gbrS.firstBox      = false;
        // gbrS.cB.minGlue    += gbrS.tmpMinGlue;
        // gbrS.cB.optGlue    += gbrS.tmpOptGlue;
        // gbrS.cB.maxGlue    += gbrS.tmpMaxGlue;
        // gbrS.tmpMinGlue    = gbrS.tmpOptGlue = gbrS.tmpMaxGlue = 0;
        // gbrS.cB.trailingSpaces = 0;
    }

    gbrS.fNoStartline = false;
    gbrS.fLastHangable = 0;

    gbrS.cB.curBox += gbrS.cB.theChRec->escapement;

    gbrS.cB.chCount++;
    gbrS.cB.streamCount++;
    gbrS.cB.theChRec++;

    BRKSetCandBreak( eHardHyphBreak );

    return in_line;
}

/*****/
/* start the stream */

static void BRKDropCapControl( MicroPoint lineOrg,
                              MicroPoint baseline )
{
    int visible      = CTIsDropCapable( gbrS.cB.theChRec->character )
                      && scCachedStyle::GetParaStyle().GetFlowdir().IsHorizontal();
    int flushleft    = gbrS.effectiveRag & (int)eRagRight;
    if ( visible && flushleft && ::DCCompute( gbrS.dcInfo,
                                              gbrS.pspec_,
                                              gbrS.cB.spec,
                                              lineOrg,
                                              baseline,
                                              gbrS.cB.theChRec->character ) ) {
        gbrS.cB.theChRec->flags.SetDropCap();
        gbrS.dcSet = true;
        gbrS.cB.streamCount++;
        gbrS.cB.theChRec++;
    }
    else {
        gbrS.dcSet = false;
        SCmemset( &gbrS.dcInfo, 0, sizeof( DropCapInfo ) );
        gbrS.cB.theChRec->flags.ClrDropCap();
    }
}

```

```

}
}

/* *****
/* we have hit end of stream, check to see if we have exceeded measure,
/* if not longjmp out, otherwise back up to a reasonable break point
/* and get out
*/

static eBreakEvent bmBRKEndStream( )
{
    MicroPoint adjustableSpace
        = gbrS.desiredMeasure - gbrS.cB.curBox;

    if ( gbrS.cB.maxGlue > adjustableSpace ) {
        BRKSetCandBreak( eEndStreamBreak );
        if ( gbrS.cB.minGlue > adjustableSpace ) {
            BRKLineDecision( 0 );
            return BRKExitLoop( );
        }
    }

    if ( gbrS.cB.optGlue > adjustableSpace ) {
        BRKSetCandBreak( eEndStreamBreak );
        BRKLineDecision( 0 );
        return BRKExitLoop( );
    }
    BRKSetCandBreak( eEndStreamBreak );

    if ( BRKLineDecision( 0 ) == scEndStream )
        return end_of_stream_reached;

    return measure_exceeded;
}

/* *****
/* This sets up the linebreaker by initing the spec, performing indents,
/* rag zone control, etc. returns the desired measure of the line
*/

static MicroPoint BRKRagControl( CharRecordP    chRec,
                                MicroPoint      x,
                                MicroPoint      y,
                                MicroPoint      measure,
                                TypeSpec        spec,
                                ushort           lineCount,
                                short            linesHyphenated )
{
    MicroPoint  dcLeftOffset;
    MicroPoint  dMeasure;

    scCachedStyle::GetCachedStyle( spec );
    gbrS.effectiveRag = scCachedStyle::GetParaStyle().GetRag();
    gbrS.brkRightMargin = scCachedStyle::GetParaStyle().GetRightBlockIndent();
    gbrS.theLineOrg = x;

    // CONSECUTIVE HYPHENATED LINE CONTROL
    if ( scCachedStyle::GetParaStyle().GetHyphenate() && linesHyphenated < scCachedStyle::GetParaStyle().GetMaxConsHyphs() )
        gbrS.allowHyphens = true;
    else
        gbrS.allowHyphens = false;

    gbrS.pspec_ = scCachedStyle::GetParaStyle().GetSpec();

    // DROP CAP CONTROL
    if ( lineCount == 0 ) {
        gbrS.cB.spec = ::BRKUpdateSpec( gbrS.theSpecRec );
        gbrS.theSpecRec++;
        ::BRKDropCapControl( x, y );
    }
}

```

```

    gbrS.theBreakColH = ggcS.theActiveColH;
    gbrS.dcLastBaseline = LONG_MIN;
}
else
    gbrS.dcSet = false;

dcLeftOffset = 0;
if ( gbrS.dcInfo.dcLineOrgChange && gbrS.dcLastBaseline != y ) {
    if ( y > gbrS.dcInfo.dcVMax || ggcS.theActiveColH != gbrS.theBreakColH )
        gbrS.dcInfo.dcLineOrgChange = 0;
    else
        /* need to compute left indent for drop caps */
        dcLeftOffset = gbrS.dcInfo.dcLineOrgChange - x;
}

// INDENTATION CONTROL
if ( gbrS.effectiveRag & (int)eRagRight && (long)lineCount < scCachedStyle::GetParaStyle().GetLinesToIndent() )
    gbrS.brkLeftMargin = scCachedStyle::GetParaStyle().GetIndentAmount() + dcLeftOffset;
else
    gbrS.brkLeftMargin = dcLeftOffset;

if ( gbrS.dcLastBaseline != y && x <= gbrS.charIndent )
    gbrS.brkLeftMargin += ( gbrS.charIndent - x );

// HANGING PUNCTUATION CONTROL
// this computes the actual overhang
if ( ( gbrS.effectiveRag & (int)eFlushLeft )
    && ( gbrS.effectiveRag & (int)eHangPuncLeft )
    && CTIsPunc( chRec->character ) )
    gbrS.brkLeftMargin -= scCachedStyle::GetParaStyle().GetLeftHangValue( chRec->character );

gbrS.theBreakColH = ggcS.theActiveColH;
gbrS.dcLastBaseline = y;

// compute the desired measure
dMeasure = measure - gbrS.brkLeftMargin - gbrS.brkRightMargin;

// compute the hyphenation zone
if ( scCachedStyle::GetParaStyle().GetRagZone() > dMeasure )
    gbrS.hyphenationZone = dMeasure / 2;
else
    gbrS.hyphenationZone = scCachedStyle::GetParaStyle().GetRagZone();

/* A LITTLE BULLET PROOFING
 * NOTE: it will be so out of whack the user will spot it fast
 */
if ( dMeasure < 0 ) {
    SysBeep(10);
    return 0;
    return one_point;
}

return dMeasure;
}

/*****

static TypeSpec BRKUpdateSpec( scSpecRecord *specRecEntry )
{
    TypeSpec    theSpec    = specRecEntry->spec();
    size_t      mlvIndex;

    scCachedStyle::GetCachedStyle( theSpec );

    /* this is to take care of the rag setting on a line */
    if ( gbrS.cb.startCount == ( gbrS.cb.streamCount - 1 ) )
        gbrS.effectiveRag = scCachedStyle::GetParaStyle().GetRag();
}

```

```

    if ( gbrS.cB.lineVal + 1 < MAXLEADVALS ) {
        gbrS.cB.specChanged++;
        mlvIndex = gbrS.cB.lineVal;

        gbrS.fMaxLineVals[mlvIndex].fSpecRec = specRecEntry;
        gbrS.fMaxLineVals[mlvIndex].fMaxLead.Set( scCachedStyle::GetCurrentCache().GetComputedLead()
    );

        gbrS.fMaxLineVals[mlvIndex].fMaxInkExtents = scCachedStyle::GetCurrentCache().GetInkExtents(

    );

        gbrS.fMaxLineVals[mlvIndex++].fOblique = scCachedStyle::GetCurrentCache().GetHorzOblique();
        gbrS.cB.lineVal = mlvIndex;

        *( gbrS.fMaxLineVals + gbrS.cB.lineVal ) = gbrS.fZeroMaxLineVals;
        gbrS.cB.specRec = specRecEntry;
    }

    return theSpec;
}

/*****
/* find the last non-space character on the line, given that what is passed in
* is the last character on the line
*/

static CharRecordP BRKLastCharOnLine( CharRecordP tmpChRec )
{
    for ( ; CTIsSpace( tmpChRec->character ); tmpChRec-- )
        ;
    return tmpChRec;
}

/*****
static void BRKRepairLastSpace( CharRecordP tmpChRec,
                                long          numberToNull )
{
    switch ( (tmpChRec-1)->character ) {
        case scQuadCenter:
            gbrS.effectiveRag = eRagCentered;
            tmpChRec -= 2;
            break;
        case scQuadLeft:
            gbrS.effectiveRag = eRagRight;
            tmpChRec -= 2;
            break;
        case scQuadRight:
            gbrS.effectiveRag = eRagLeft;
            tmpChRec -= 2;
            break;
        case scQuadJustify:
            gbrS.effectiveRag = eRagJustified;
            tmpChRec -= 2;
            break;
        case scHardReturn:
        case scVertTab:
            tmpChRec -= 2;
            break;
        default:
            tmpChRec--;
            break;
    }

    gbrS.totalTrailingSpace = 0;
    for ( ; numberToNull && tmpChRec->character == scWordSpace; tmpChRec--, numberToNull-- ) {
        if ( gHiliteSpaces )
            gbrS.totalTrailingSpace += tmpChRec->escapement;
    }

    scAssert( !numberToNull );
}

```

```

}

/*****/

static void BRKAdjustWordSpace( CharRecordP prevChar,
                                GlyphSize  adjustment,
                                long        numSpaces,
                                long        endSpaces )
{
    /* when we come in here prevchar points to the first word of the next line
     * we need to ignore it if it is a wordspace
     */
    if ( prevChar->character == scWordSpace )
        prevChar--;

    for ( ; endSpaces && prevChar > gbrS.gStartRec; prevChar-- ) {
        if ( prevChar->character == scWordSpace )
            endSpaces--;
    }
    scAssert( endSpaces == 0 );
    for ( ; numSpaces && prevChar >= gbrS.gStartRec; prevChar-- ) {
        if ( prevChar->character == scWordSpace ) {
            prevChar->escapement = adjustment;
            numSpaces--;
        }
    }
    scAssert( numSpaces == 0 );
}

/*****/

static void BRKRepairFinalSpace( )
{
    scAssert( gbrS.cb.theChRec->character == 0 );
    BRKRepairLastSpace( gbrS.cb.theChRec, gbrS.cb.trailingSpaces );
}

/*****/

BreakStruct::BreakStruct()
{
}

/*****/

BreakStruct::~~BreakStruct()
{
}

/*****/

void BreakStruct::Init()
{
    pspec_.clear();
    cb.Init();
    for ( int i = 0; i < MAXBREAKVALS; i++ )
        gbrS.candBreak[i].Init();
}

/*****/
/* free the memory associated with the breaking machine */

void BRKFreeMach( )
{
    delete [] gbrS.breakMach,          gbrS.breakMach      = 0;
    delete [] gbrS.fMaxLineVals,       gbrS.fMaxLineVals  = 0;
    delete [] gbrS.candBreak,          gbrS.candBreak      = 0;
}

/*****/
/* init the breaking machine */

void BRKInitMach( )

```

```

{
    int i;

    gbrS.breakMach      = new BrFunc[ SIZE_OF_MACHINE ];
    gbrS.fMaxLineVals   = new scMaxLineVals[ MAXLEADVALS ];
    gbrS.candBreak      = new CandBreak[ MAXBREAKVALS ];

    for ( i = 0; i < SIZE_OF_MACHINE; i++ ) {
        switch ( i ) {
            case scTabSpace:
                gbrS.breakMach[i] = bmBRKTab;
                break;
            case scWordSpace:
                gbrS.breakMach[i] = bmBRKWordSpace;
                break;
            case scEndStream:
                gbrS.breakMach[i] = bmBRKEndStream;
                break;
            case scEnDash:
            case scEmDash:
            case scBreakingHyphen:
            case '=':
                gbrS.breakMach[i] = bmBRKHyphen;
                break;
            case scFillSpace:
                gbrS.breakMach[i] = bmBRKFillSpace;
                break;
            case scRulePH:
                gbrS.breakMach[i] = bmBRKRule;
                break;
            case scFixAbsSpace:
            case scFigureSpace:
            case scThinSpace:
            case scEnSpace:
            case scEmSpace:
                gbrS.breakMach[i] = bmBRKFixSpace;
                break;
            case scFixRelSpace:
                gbrS.breakMach[i] = bmBRKRelSpace;
                break;
            case scVertTab:
                gbrS.breakMach[i] = bmBRKVertTab;
                break;
            case scQuadCenter:
            case scQuadLeft:
            case scQuadRight:
            case scQuadJustify:
                gbrS.breakMach[i] = bmBRKQuad;
                break;
            case scHardReturn:
                gbrS.breakMach[i] = bmBRKHardReturn;
                break;
            case scField:
                gbrS.breakMach[i] = bmBRKField;
                break;
            default:
                gbrS.breakMach[i] = bmBRKChar;
                break;
        }
    }
}

/*****/

static Bool BRKStillMoreChars( CharRecordP  chRec,
                               long          count )
{
    for ( ; count--; chRec++ ) {
        if ( CTIsVisible( chRec->character ) )
            return true;
    }
    return false;
}

```

```

/*****/

Bool BRKJustify( CharRecordP   chRec,      /* the character array */
                long   start,      /* count into ch array to start the linebreak */
                long   stop,      /* count into ch array of end of line */
                MicroPoint measure ) /* measure to justify to */
{
    long   spaces,
          count;
    MicroPoint delta;
    MicroPoint boxWidth;
    CharRecordP holdChRec;
    Bool   changed = false;

    chRec   += start;
    holdChRec = chRec;

    boxWidth = 0;
    for ( spaces = 0, count = stop - start; count; chRec++, count-- ) {
        switch ( chRec->character ) {
            case scWordSpace:
                if ( BRKStillMoreChars( chRec, (long)count ) )
                    spaces++;
                break;
            default:
                boxWidth += chRec->escapement;
                break;
        }
    }

    delta = measure - boxWidth;
    if ( spaces ) {
        delta = scRoundMP((REAL)delta / spaces);

        for ( chRec=holdChRec, count = stop-start; count; chRec++,count-- ) {
            switch ( chRec->character ) {
                case scWordSpace:
                    if ( spaces ) {
                        spaces--;
                        if ( !changed && chRec->escapement != delta )
                            changed = true;
                        chRec->escapement = (GlyphSize)delta;
                    }
                    break;
                default:
                    break;
            }
        }
    }
    return changed;
}

```

```

/*****/

#if 0
static void BRKCharJapanese( )
{
    MicroPoint   adjustableSpace;
    UCS2         theCharacter;
    CharBits     cb;
    Bool         noStartline,
                noEndline;

    adjustableSpace = gbrS.desiredMeasure - gbrS.cb.curBox;
    theCharacter    = gbrS.cb.theChRec->character;

    cb = TSCharBits( scCachedStyle::GetCurrentCache().fmTheSpec, theCharacter );

    if ( cb.fTheBits.fCharClass )
        noStartline = true;
}

```

```

else
    noStartline = false;

if ( cb.fTheBits.fCharClass )
    noEndline = true;
else
    noEndline = false;

// ValidateBits( theCharacter, cb );

if ( gbrS.numTargetChars > 0 ) {    /* inhibit breaks in      */
    gbrS.numTargetChars--;          /* target sequence      */
}
else {
    /* set a potential break before every character */
    if ( !( noStartline || gbrS.fNoStartline ) ) {
        BRKSetCandBreak( eCharBreak );

        if ( BRKExceedVals( adjustableSpace ) ) {
            BRKLineDecision( 0 );
            return BRKExitLoop( );
        }
    }

    if ( gbrS.firstBox )
        BRKSetFirstBox( );
}

gbrS.cb.curBox += gbrS.cb.theChRec->escapement;

if ( noEndline || ( !scCachedStyle::GetCurrentCache().fmBreakableNumbers && cb.fTheBits.fDigit ) )
{
    gbrS.fNoStartline = true;
}
else
    gbrS.fNoStartline = false;

if ( cb.fTheBits.fHangable )
    gbrS.cb.fHangable = gbrS.cb.theChRec->escapement;
else
    gbrS.cb.fHangable = 0;

gbrS.cb.chCount++;
gbrS.cb.streamCount++;
gbrS.cb.theChRec++;
}
#endif

/*****/

CandBreak& CandBreak::operator=( const CandBreak& cb )
{
    breakCount      = cb.breakCount;
    startCount      = cb.startCount;
    streamCount      = cb.streamCount;
    wsSpaceCount     = cb.wsSpaceCount;
    spaceCount       = cb.spaceCount;
    trailingSpaces   = cb.trailingSpaces;
    chCount          = cb.chCount;
    fillSpCount      = cb.fillSpCount;
    lineVal          = cb.lineVal;
    breakVal         = cb.breakVal;
    minGlue          = cb.minGlue;
    optGlue          = cb.optGlue;
    maxGlue          = cb.maxGlue;
    curBox           = cb.curBox;
    fHangable        = cb.fHangable;
    theChRec         = cb.theChRec;
    specChanged      = cb.specChanged;
    spec             = cb.spec;
    specRec          = cb.specRec;
}

```


/#####/

[illegible]

[illegible]

[illegible]

File: sccallbk.h

\$Header: /Projects/Toolbox/ct/SCCALLBK.H 2 5/30/97 8:45a Wmanis \$

Contains: The call backs to the client from the composition toolbox.

Written by: Manis

Copyright (c) 1989-1994 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

@doc

```

*****/

#ifdef _H_SCSPECSY
#define _H_SCSPECSY

#ifdef SCMACINTOSH
#pragma once
#endif

#include "sctypes.h"

//-----
// The following are call backs that the application must support
// in order for the above selection messages to work properly.
//
// @CALLBACK Provides the Toolbox with the drawing
// context of the column, used for highlighting or drawing.
//
status scIMPL_IMPORT APPDrawContext(
    APPColumn      appCol,      // @parm <t APPColumn>
    const scColumn* col,        // @parm <c scColumn>
    APPDrwCtx&     drwctx );    // @parm <t APPDrwCtx>

// CALL BACK - this informs the composition toolbox whether it should recompose
// this column or not, the client may prevent recomposition of columns that are not
// visible, though keep in mind that if a subsequent column is visible we
// will have to recompose this column at some point in time.
Bool scIMPL_IMPORT APPRecomposeColumn( APPColumn );

/* ----- */
// @CALLBACK Maps a pointer of a client object to an ID on disk. Typically
// a TypeSpec.

enum stDiskidClass {
    diskidUnknown,
    diskidColumn,
    diskidTypespec,
    diskidOther
};

long scIMPL_IMPORT APPPointerToDiskID(
    APPCtxPtr,
    void* clientObj,      // @parm Pointer to client object.
    stDiskidClass );      // class of object

// @CALLBACK Maps a disk ID to a pointer. Typically a TypeSpec.

```

```

void* scIMPL_IMPORT      APPDiskIDToPointer(
                          APPCtxPtr,
                          long    diskID,      // @parm A value returned by <f APPPointerToDiskID>
                                              // that we want a valid pointer to now.
                          stDiskidClass );     // class of object

```

```

/* ===== */

```

```

//
// called periodically by the Toolbox during actions
// that will take some time. If the call for an event returns 0,
// the action will be aborted and control will revert to application.
// The client can give the Toolbox and hint as to how much more time
// it can process for. The client can return a negative number as
// an indicator to get out fast.
//

```

```

// this describes the current process type that the toolbox
// is performing.

```

```

typedef enum scProcTypes {
    scDrawProc,      // toolbox is drawing
    scReformatProc   // toolbox is reformatting
} scProcType;

```

```

scTicks scIMPL_IMPORT  APPEventAvail( scProcType );

```

```

/* ===== */

```

```

// @enum eSpecChange | When a TypeSpec is changed externally to the
// Toolbox, the Toolbox needs to be informed that a change has occurred
// so that reformatting may occur. In an effort to minimize the work
// the function <f SpecTaskCalculate> can calculate the minimum amount
// of work that needs to be done. (e.g. changing the color of a spec
// should only require repainting and not reformatting,
// SpecTaskCalculate(scColor) would return eSCRepaint ) With the return
// value of SpecTaskCalculate one can inform the Toolbox about the changed
// spec <f SCENG_ChangedTS>( ts, <t eSpecTask>, <c scRedisplist> ) and
// get information about the minimal area to update.

```

```

typedef enum eSpecChanges {

```

```

    scLanguage,
    scFont,
    scColor,
    scRenderAttribute,
    scCharTransform,

```

```

    scPointSize,
    scSetSize,
    scHoblique,
    scVoblique,
    scRotation,

```

```

    scKern,
    scMarginKern,
    scTrack,
    scMinLsp,
    scOptLsp,
    scMaxLsp,

```

```

    scMinWsp,
    scOptWsp,
    scMaxWsp,

```

```

    scLead,
    scBaseline,
    scAboveLead,
    scBelowLead,

```

```

    scIndLines,
    scIndAmount,
    scIndDepth,
    scIndLeftBL,

```

```

scIndRightBL,      scIndentExtra1, scIndentExtra2,
scNoHyphLastWord,
scColNoBreak,
scKeepNext,
scLinesBefore,
scLinesAfter,      scWidowOrphanExtra1, scWidowOrphanExtra2,

scRag,
scForceJust,
scRagPattern,
scRagZone,
scKernMargins,
schLeft,
schRight,
schLeftAmount,
schRightAmount,    scRagExtra1, scRagExtra2, schPuncExtra1, schPuncExtra2,

schHyphenation,
schHyphChar,
schHyphLines,
schHyphExcep,
schHyphMinSize,
schPreHyphs,
schPostHyphs,
schHyphPropensity,
schHyphCaps,
schHyphAcros,      schHyphExtra1, schHyphExtra2,

scDCShow,
scDCChar,
scDCPtSize,
scDCsetSize,
scDChOffset,
scDCvOffset,
scDChBorder,
scDCvBorder,
scDCfont,
scDCcolor,

scMaxFillChars,
scFillPos,
scFillChar,
scFillAlign,

scMaxTabs,
scTabPos,
scTabAlign,
scTabChar,
scTabFillAlign,

scMinMeasure,
scRunAroundBorder,
scFirstLine,

scMaxValType
} eSpecChange;

```

```
// @CALLBACK Used to determine minimal work on a spec change.
```

```
// @rdesc <t eSpecTask>
```

```
eSpecTask      SpecTaskCalculate(
                eSpecChange specChange ); // @parm <t eSpecChange>
```

```

/* ===== */
/* ===== */
/* ===== SPEC SUB-YSTEM CALL BACKS ===== */
/* ===== */
/* ===== */

```

```
class scStyle;
```

```
// @CALLBACK Gets the scStyle structure.
```

```

status scIMPL_IMPORT    TGetStyle( TypeSpec& ts,          // @parm <t TypeSpec>
                                   scStyle& style );    // @parm <c scStyle>

// @CALLBACK This call back is used to determine positioning of tabs.
// @ex Default value for tab positioning might be. |
// tabInfo.xPos = ( xPos / defaultTabWidth + 1 ) * defaultTabWidth );
//
status scIMPL_IMPORT    TTabInfo(
                        TypeSpec& paraspec,          // paragraph spec
                        TypeSpec& ts,                // @parm <t TypeSpec>
                        scTabInfo& tabInfo,          // @parm <t scTabInfo>
                        MicroPoint xPos,             // @parm X position in column.
                        MicroPoint yPos,             // @parm Depth in column.
                        long lineNum );              // @parm Line num in para.

// default wordspace
status scIMPL_IMPORT    TSfillCharInfo( TypeSpec&,
                                         UCS2&,
                                         eFCAlignment&,
                                         MicroPoint,
                                         MicroPoint,
                                         long );

// default return false
Bool scIMPL_IMPORT    TDropCap( TypeSpec&,          // para spec
                                TypeSpec&,          // character spec
                                DCPosition&,         // position struct
                                UCS2 );              // dropcap character
//line Bool          TDropCap( TypeSpec, DCPosition& ) { return false; }

/* ===== */
/* ===== */
// COLUMN SPECIFICATIONS - 'CS'

// By sending in the two specs the spec management system may generate
// a value intelligently, either a hard coded value or parametrically
// derived value using the pointsize of the type

// @CALLBACK Position of first line in a column, default should
// be point size, this is not for use with dropcaps. Client
// may return any reasonable value and may use none, one or
// both of the parameters.
MicroPoint scIMPL_IMPORT    CSfirstLinePosition(
                        APPColumn appcol,          // @parm <t APPColumn>
                        TypeSpec ts );              // @parm <t TypeSpec>

// @CALLBACK Position of last line in a column,
// default should be zero since this will allow
// multiple columns with different point sizes
// to bottom align.
MicroPoint scIMPL_IMPORT    CSlastLinePosition(
                        APPColumn appcol,          // @parm <t APPColumn>
                        TypeSpec ts );              // @parm <t TypeSpec>

// @CALLBACK Border to inset text from shape applied to
// column -- default is 0, the spec is the first
// encountered in the column.
inline MicroPoint    CSrunaroundBorder(
                        APPColumn appcol,          // @parm <t APPColumn>
                        TypeSpec ts )              // @parm <t TypeSpec>
{ return 0; }

/* ===== */
/* ===== FONT METRIC CALL BACKS ===== */
/* ===== */

////////// DESIGN METRICS //////////
// DESIGN COORDINATES ARE THE RELATIVE UNIT SYSTEM DEFINED
// IN scBaseRLUSystem

// @CALLBACK Return the escapement of the glyph in design coordinates.
//

```

```

RLU scIMPL_IMPORT  FigetRLUEScapement(
    const scFontRender& fr,      // @parm <t scFontRender>
    UCS2  ch );                 // @parm Glyph.

RLU scIMPL_IMPORT  FigetRLUEScapement( const scFontRender&,
    UCS2,
    RLU /*suggestedWidth*/ );

// @CALLBACK Return the kerning value of the glyphs in design coordinates.
//
RLU scIMPL_IMPORT  FigetRLUKern(
    const scFontRender& fr,      // @parm <t scFontRender>
    UCS2  ch1,                  // @parm Glyph one.
    UCS2  ch2 );                // @parm Glyph two.

// @CALLBACK Return the glyph ink box in design coordinates
//
scRLURect& scIMPL_IMPORT  FigetRLUExtents(
    const scFontRender& fr,      // @parm <t scFontRender>
    UCS2  ch,                  // @parm Glyph one.
    scRLURect&      inkBox );  // @parm <c scRLURect>

// @CALLBACK Return the various font metrics in design coordinates
void scIMPL_IMPORT  FigetRLUFontExtents(
    const scFontRender& fontrender, // @parm <t scFontRender>
    RLU& capHite, // @parm Cap height.
    RLU& xHite, // @parm Lower case x height.
    RLU& ascenderHite, // @parm Ascender height.
    RLU& descenderDepth, // @parm Descender height.
    scRLURect& maxInkExt ); // @parm <c scRLURect> union of
                                // ink extents of all glyphs in
                                // font.

////////// DEVICE METRICS //////////

// @CALLBACK Return the escapement of the glyph in device coordinates
// ( transformed into toolbox coordinates ).
GlyphSize scIMPL_IMPORT  FigetDEVEscapement(
    const scFontRender& fr,      // @parm <t scFontRender>
    UCS2  ch );                 // @parm Glyph.

GlyphSize scIMPL_IMPORT  FigetDEVEscapement( const scFontRender&,
    UCS2,
    GlyphSize /*suggestedWidth*/ );

// @CALLBACK Return the kerning value of the glyphs in device coordinates
// ( transformed into toolbox coordinates ).
GlyphSize scIMPL_IMPORT  FigetDEVKern(
    const scFontRender& fr,      // @parm <t scFontRender>
    UCS2  ch1,                  // @parm Glyph one.
    UCS2  ch2 );                // @parm Glyph two.

// @CALLBACK Return the glyph ink box in device coordinates
// ( transformed into toolbox coordinates ).
//
scXRect& scIMPL_IMPORT  FigetDEVExtents(
    const scFontRender& fr,      // @parm <t scFontRender>
    UCS2  ch,                  // @parm Glyph one.
    scXRect&      inkBox );    // @parm <c scXRect>

// @CALLBACK Return the various font metrics in device coordinates
// ( transformed into toolbox coordinates ).
void scIMPL_IMPORT  FigetDEVFontExtents(
    const scFontRender& fontrender, // @parm <t scFontRender>
    MicroPoint& capHite, // @parm Cap height.
    MicroPoint& xHite, // @parm Lower case x height.
    MicroPoint& ascenderHite, // @parm Ascender height.
    MicroPoint& descenderDepth, // @parm Descender height.
    scXRect& maxInkExt ); // @parm <c scXRect> union of

```



```
// ink extents of all glyphs in
// font.
```

```
/* ===== */
/* -----  HYPHENATION SUB-SYSTEM -----  */
/* ===== */

// @CALLBACK  Initializes the Hyphenation sub-system to the indicated language.
// Returns true if language properly initied.
//
Bool scIMPL_IMPORT  HYFLanguageInit(
    APPLanguage lang );          // @parm <t APPLanguage>

// @CALLBACK  Chars are in word, NULL terminated, return hyph values in hyfs, max
// len of either is 64. if word is hyphenated return true.
//
Bool scIMPL_IMPORT  HYFWord(
    const UCS2* theWord,          // @parm The word.
    short*      hyphArray );      // @parm The hyphenation array.

/* ===== */
/* -----  CHAR DRAWING CALLBACKS -----  */
/* ===== */

// @CALLBACK  Called before the start of drawing a line.
//
void scIMPL_IMPORT  APPDrawStartLine(
    APPDrwCtx      drwctx,          // @parm <t APPDrawCtx>
    MicroPoint     x,                // @parm X origin of line.
    MicroPoint     y,                // @parm Y origin of line.
    const scXRect& inkext );         // @parm Max ink extents of line.

// @CALLBACK  Called n times ( for each style or full buffer ) between a APPDrawStartLine
// and an APPDrawEndLine.
// @xref <f SCCOL_Update>
void scIMPL_IMPORT  APPDrawString(
    APPDrwCtx      dc,              // @parm Pass thru context.
    const scGlyphArray* ga,         // @parm <t scGlyphArray> array.
    short          num,             // @parm Number of glyphs in array.
    MicroPoint     x,              // @parm X origin of string.
    MicroPoint     y,              // @parm Y origin of string.
    const scGlyphInfo& gi );        // @parm <t scGlyphInfo>

// @CALLBACK  Called at the end of drawing a line.
void scIMPL_IMPORT  APPDrawEndLine(
    APPDrwCtx dc );                // @parm <t APPDrwCtx> drawing context.

// @CALLBACK Used to draw hiliting rectangles.
void scIMPL_IMPORT  APPDrawRect(
    const scXRect& xorRect,         // @parm <c scXRect> to xor.
    APPDrwCtx      dc,              // @parm <t APPDrwCtx> drawing context.
    Bool           sliverCursor );

void scIMPL_IMPORT  APPDrawRule( const scMuPoint&,
    const scMuPoint&,
    const scGlyphInfo&,
    APPDrwCtx );

/* ===== */
/* -----  -----  */
/* ===== */

class clField {
public:
    static clField&      createField( scStream*, uint8 );

    virtual uint8        id() const = 0;
};
```

[illegible]

```
#endif /* _H_SCCALLBK */
```

File: SC-SpecChng.c

SHheader: /Projects/Toolbox/ct/SC_SPCHG.CPP 2 5/30/97 8:45a Wmanis S

Contains:

When type specs change their are certain types of things that need to be done to bring the world back into equilibrium. These tasks typically involve REFORMATTING and REPAINT. Since a certain number of the formatting computations are held with the characters themselves the reformatting requires two operations. We will call these RETABULATION - correcting the escapement stored with the characters - and the LINEBREAKING - the act of breaking text into lines.

Therefore when a spec changes one or more tasks may need to be performed, we want to determine the minimum set of tasks to perform to return the world to equilibrium.

The tasks are performed in the following order:

TABULATION
LINE BREAKING
PAINTING

Here a few examples of spec changes and what they should cause:

color change	- scREPAINT
word space change	- scREBREAK & scREPAINT
lead change	- scREBREAK & scREPAINT
font change	- scRETABULATE, scREBREAK & scREPAINT
pointsize change	- scRETABULATE, scREBREAK & scREPAINT
setsize change	- scRETABULATE, scREBREAK & scREPAINT
pair/track kerning change	- scRETABULATE, scREBREAK & scREPAINT
hyphenation language change	- scRETABULATE, scREBREAK & scREPAINT
# of consecutive hyph change	- scREBREAK & scREPAINT

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

*****/
#include "sccallbk.h"

```
eSpecTask  SpecTaskCalculate( eSpecChange changeType )
{
    switch ( changeType ) {
        case scHoblique:
        case scVoblique:
            return (eSpecTask)((int)eSCRetabulate | (int)eSCRepaint);

        case scLanguage:
        case scFont:
        case scCharTransform:
        case scPointSize:
        case scSetSize:
        case scRotation:
        case scKern:
        case scTrack:
        case scMinLsp:
```

```

case scOptLsp:
case scMaxLsp:
case scMinWsp:
case scOptWsp:
case scMaxWsp:
case scHyphenation:
case scHyphLines:
case scHyphExcep:
case scHyphMinSize:
case scPreHyphs:
case scPostHyphs:
case scHyphPropensity:
case scHyphCaps:
case scHyphAcros:
case scHyphExtra1:
case scHyphExtra2:
default:
    return eSCRetabulate;

```

```

case scColor:
case scRenderAttribute:
// case scULShow:
// case scULpos:
// case scULthick:
// return eSCRepaint;

```

```

case scLead:
case scBaseline:
case scAboveLead:
case scBelowLead:
case scIndLines:
case scIndAmount:
case scIndDepth:
case scIndLeftBL:
case scIndRightBL:
case scIndentExtra1:
case scIndentExtra2:
case scColNoBreak:
case scKeepNext:
case scLinesBefore:
case scLinesAfter:
case scWidowOrphanExtra1:
case scWidowOrphanExtra2:
case scRag:
case scForceJust:
case scRagPattern:
case scRagZone:
case scKernMargins:
case scHLeft:
case scHRight:
case scHLeftAmount:
case scHRightAmount:
case scRagExtra1:
case scRagExtra2:
case scHPuncExtra1:
case scHPuncExtra2:
case scDCShow:
case scDCptSize:
case scDCsetSize:
case scDCchOffset:
case scDCcvOffset:
case scDCchBorder:
case scDCcvBorder:
case scDCfont:
case scDCcolor:
case scMaxFillChars:
case scFillPos:
case scFillChar:
case scFillAlign:
case scMaxTabs:
case scTabPos:
case scTabAlign:
case scTabChar:

```

} }

[illegible][illegible]

File: SC_SYSCO.C

\$Header: /Projects/Toolbox/ct/SC_SYSCO.CPP 2 5/30/97 8:45a Wmanis \$

Contains: Implementation of transfer of clipboard data
to external format.

Written by: Lucas

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

*****/

/* THESE ARE STUBS AND ARE BY NO MEANS COMPLETE OR ROBUST */

```
#include "sctypes.h"
#ifdef SCMACINTOSH
#include <Memory.h>
#endif

/* ===== */
SystemMemoryObject::SystemMemoryObject()
{
#ifdef SCWINDOWS
    fSYSHandle = GlobalAlloc( 0, GPTR ); // since GHND allows only 64k bytes
#elif defined( SCMACINTOSH )
    fSYSHandle = NewHandle( 0 );
#endif
}

/* ===== */
SystemMemoryObject::~SystemMemoryObject()
{
#ifdef SCWINDOWS
    if ( fSYSHandle )
        GlobalFree( fSYSHandle );
#elif defined( SCMACINTOSH )
    if ( fSYSHandle )
        DisposHandle( fSYSHandle );
#endif
}

/* ===== */
void SystemMemoryObject::ReleaseMem()
{
    fSYSHandle = 0;
}

/* ===== */
long SystemMemoryObject::HandleSize( void )
{
#ifdef SCWINDOWS
    return (size_t)GlobalSize( fSYSHandle );
#elif defined( SCMACINTOSH )
    return GetHandleSize( fSYSHandle );
#endif
}

/* ===== */
```

```

status SystemMemoryObject::SetHandleSize( long newsize )
{
    #if defined( SCWINDOWS )
        fSYSHandle = GlobalReAlloc( fSYSHandle, newsize, GMEM_MOVEABLE | GMEM_ZEROINIT );
        return fSYSHandle != 0 ? scSuccess : scERRmem;
    #elif defined( SCMACINTOSH )
        ::SetHandleSize( fSYSHandle, newsize );
        return MemError() == noErr ? scSuccess : scERRmem;
    #endif
}

/* ===== */

void *SystemMemoryObject::LockHandle( void )
{
    #if defined( SCWINDOWS )
        return GlobalLock( fSYSHandle );
    #elif defined( SCMACINTOSH )
        HLock( fSYSHandle );
        return *fSYSHandle;
    #endif
}

/* ===== */

void SystemMemoryObject::UnlockHandle( void )
{
    #if defined( SCWINDOWS )
        GlobalUnlock( fSYSHandle );
    #elif defined( SCMACINTOSH )
        HUnlock( fSYSHandle );
    #endif
}

/* ===== */

```

```
/* mapping on reading input buffer or file */
```

```
#ifndef noCMinputMap
```

```
UCS2 CMinputMap( ushort ch )
{
    return ch;
}
```

```
#endif /* noCMinputMap */
```

```
/*-----/
```

```
#ifndef noCMmakeKeyRecordTwo
```

```
void CMmakeKeyRecordTwo( scKeyRecord&      keyRecord,
                        UCS2                keyCode,
                        GlyphSize           val,
                        TypeSpec            spec,
                        Bool                restoreSelection,
                        scStreamLocation&    mark )
{
    keyRecord.keycode()      = keyCode;
    keyRecord.replacedchar() = 0;
    keyRecord.escapement()   = val;
    keyRecord.spec()         = spec;
    keyRecord.noop()         = false;
    keyRecord.restoreselect() = restoreSelection;
    keyRecord.mark()         = mark;
}
```

```
#endif /* noCMmakeKeyRecordTwo */
```

```
/*-----/
```


File: SC_UTLWI.C

\$Header: /Projects/Toolbox/ct/SC_UTLWI.CPP 2 5/30/97 8:45a Wmanis \$

Contains: WINDOWS versions of low level debugging stuff

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

```
*****/
#include "sctypes.h"
#include "scexcept.h"
```

```
/*=====*/
void SCDebugStr ( const scChar* cstr )
```

```
{
    OutputDebugString( cstr );
```

```
-----*/
void SCAssertFailed ( const scChar* assertStr,
                     const scChar* file,
                     int          lineNum )
```

```
{
    scChar buf[256];
```

```
    if ( scStrlen( assertStr ) + scStrlen( file ) + 4 < 256 )
        wsprintf( buf, scString( "ASSERT FAILED \"%s\" file \"%s\" line %d\n" ),
                  assertStr, file, lineNum );
```

```
    else
        scStrcpy( buf, scString( "ASSERT STRING TOO LONG!!!!\n" ) );
```

```
    SCDebugStr( buf );
```

```
    #if SCDEBUG < 1
        raise( scERRassertFailed );
```

```
    #else
        SCDebugBreak();
```

```
        // set doit to true if you want to raise an exception
        int doit = 0;
        if ( doit )
            raise( scERRassertFailed );
```

```
    #endif
}
```

```
/*=====*/
void SCDebugBreak( void )
```

```
{
    #if SCDEBUG > 1
        DebugBreak();
```

```
    #else
        #ifdef _WIN32
            Beep( 500, 100 );
        #else
            MessageBeep( -1 );
        #endif
    }
```

[illegible][illegible][illegible]

```

/*****
File:      SC_UTLTC.C

$Header: /Projects/Toolbox/ct/SC_UTMAC.CPP 2      5/30/97 8:45a Wmanis $

Contains:   Think C utilities

Written by: Manis

```

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/

// #include "scport.h"
// #include "capplia.h"
// #include "constant.h"
// #include "tbutilities.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Dialogs.h>
#include <SegLoad.h>
#include <QuickDraw.h>
#include <OSUtils.h>
#include "scTypes.h"

#ifdef THINK_CPLUS ) && THINK_CPLUS < 0x0700
    #include <pascal.h>
#else
    #include <Strings.h>
#endif

Boolean gSCUseSysBreak;

/* ===== */
void SCDebugStr( const char *str )
{
    char buf[256];

    strcpy( buf, str );

    // if ( gApplication->TestDebuggerPresence() ) {
    //     if (gSCUseSysBreak)
    //         SysBreakStr( c2pstr( buf ) );
    //     else
    //         DebugStr( c2pstr( buf ) );
    // }
    // else {
    //     ParamText( c2pstr( buf ), (StringPtr)"", (StringPtr)"", (StringPtr)"" );
    //     PositionDialog( 'ALRT', ALRTgeneral );
    //     InitCursor();
    //     Alert( ALRTgeneral, NULL );
    // }
}

/* ===== */

void SCAssertFailed( const scChar *str, const scChar *file, int line )
{
    char buf[256];

    sprintf( buf, "ASSERT FAILED \"%s\" File %s Line %ld", str, file, line );
}

```

/ * ~~~~~ * /

[illegible][illegible]

File: SCAPI.C

\$Header: /Projects/Toolbox/ct/SCAPI.CPP 3 5/30/97 8:45a Wmanis \$

Contains: Application Program Interface for the
Stonehand Composition Toolbox. For the most part
this file is simply a bottle neck module. All
documentation for the functions contained within
are found in scappint.h.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

*****/

```
#include "scappint.h"
#include "scpubobj.h"

#include "scannota.h"
#include "scapptex.h"
#include "sccolumn.h"
#include "scexcept.h"
#include "scstcach.h"
#include "scglobda.h"
#include "scmem.h"
#include "scparagr.h"
#include "scregion.h"
#include "scset.h"
#include "scstream.h"
#include "sctextli.h"

static int      gInited;
static int      gBaseError;
int             scDebugTrace = 0;

/* ===== */

#if 0
static int gInputLevel;

// if scDebugTrace is set to a value >0 all calls into
// the toolbox will be traced, may be useful for understanding
// a behavior or pointing the finger!

inline void EnterMonitor( const scChar *str )
{
    scAssert( gInited );
    SCDebugTrace( 0, scString( "\n+%s\n" ), str );
}

inline void ExitMonitor( const scChar *str )
{
    SCDebugTrace( 0, scString( "-%s\n" ), str );
}
#else
```

```

#define EnterMonitor( x )
#define ExitMonitor( x )
#endif

```

```

/* ===== */

```

```

void      BRKInitMach( void );
void      BRKFreeMach( void );

```

```

char* stoneVersion = __DATE__ " - " __TIME__;

```

```

status scIMPL_EXPORT  SCENG_Init( int baseError )
{

```

```

    // The following are pool definitions that are passed
    // to the initialization of the memory manager.
    // The last pool is the default memory allocation pool
    // all others are fixed size pools, these are not sorted
    // at this time

```

```

    static scPoolInfo  objPools[] = {
        { sizeof( scTextline ),      0 },
        { sizeof( scContUnit ),      0 },
        { sizeof( scAbstractArray ), 0 },
        { 0,                          0 }
    };
};

```

```

#ifdef useCPLUSEXCEPTIONS

```

```

    // if we are not using C++ exceptions - initialize our own
    // exception manager.

```

```

    scExceptContext::Initialize( 0 );

```

```

#endif

```

```

    status stat = scSuccess;

```

```

    gBaseError = baseError;

```

```

    try {

```

```

        MEMInit( objPools );           // initialize memory manager
        scAssert( sizeof( CharRecord ) == ( sizeof( long ) * 2 ) );
        BRKInitMach();                 // initialize breaking machine
        scCachedStyle::BuildCache( 16 ); // build internal spec cache
        gInited = true;
    }

```

```

    IGNORE_RERAISE;

```

```

    return stat;

```

```

/* ===== */

```

```

status scIMPL_EXPORT  SCENG_Fini( void )
{

```

```

    status stat = scSuccess;

```

```

    try {

```

```

        scAssert( gInited );
        BRKFreeMach();
        scColumn::FiniCTXList();
        scCachedStyle::DeleteCache();

```

```

        gInited = false;
        MEMFini();
    }

```

```

    IGNORE_RERAISE;

```

```

    return stat;

```

```

/* ===== */

```

```

status scIMPL_EXPORT  SCENG_RetainMemory ( void )

```

```

{
    // MEMSetRestrictions( memRetain );
    return scSuccess;
}

/* ===== */

status scIMPL_EXPORT    SCENG_UseRetainedMemory ( void )
{
    // MEMSetRestrictions( memUseRetained );
    return scSuccess;
}

/* ===== */

status scIMPL_EXPORT    SCENG_ReleaseMemory ( void )
{
    // MEMSetRestrictions( memNoRestrictions );
    return scSuccess;
}

/* ===== */

status scIMPL_EXPORT    SCENG_ChangedTS ( TypeSpec      ts,
                                           eSpecTask      task,
                                           scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCENG_ChangedTS" ) );

    scCachedStyle::StyleInvalidateCache ( ts );

    try {
        if ( task & eSCDoAll )
            scColumn::ChangedTS( ts, task, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCENG_ChangedTS" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCHTS_Alloc( scSpecLocList*&  cslist,
                                     scStream*          stream )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCCHTS_Alloc" ) );

    cslist = 0;

    try {
        cslist = SCNEW scSpecLocList( stream );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCHTS_Alloc" ) );

    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCHTS_Delete( scSpecLocList*& cslist )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCCHTS_Delete" ) );

    try {
        delete cslist, cslist = 0;
    }
}

```

```

    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCHTS_Delete" ) );

    return stat;
}

/* ----- */
status scIMPL_EXPORT SCTSL_Alloc( scTypeSpecList*& tslist )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCTSL_Alloc" ) );

    tslist = 0;

    try {
        tslist = SCNEW scTypeSpecList;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCTSL_Alloc" ) );

    return stat;
}

/* ----- */
status scIMPL_EXPORT SCTSL_Delete( scTypeSpecList*& tslist )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCTSL_Delete" ) );

    try {
        delete tslist, tslist = 0;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCTSL_Delete" ) );

    return stat;
}

/* ----- */
status scIMPL_EXPORT SCRDL_Alloc( scRedisList*& rdlist )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCRDL_Alloc" ) );

    rdlist = 0;

    try {
        rdlist = SCNEW scRedisList;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCRDL_Alloc" ) );

    return stat;
}

/* ----- */
status scIMPL_EXPORT SCRDL_Delete( scRedisList*& rdlist )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCRDL_Delete" ) );

```



```

    try {
        delete rdlist, rdlist = 0;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCRDL_Delete" ) );

    return stat;
}

/* ===== */
/* Recompose a single column with extreme prejudice */

status scIMPL_EXPORT    SCCOL_Recompose( scColumn*      col,
                                         scRedisList*   redisplList )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCCOL_Recompose" ) );

    try {
        col->Rebreak( redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_Recompose" ) );
    return stat;
}

/* ===== */
/* Recompose a single column with extreme prejudice */

status scIMPL_EXPORT    SCRebreakCol ( scColumn*      col,
                                       scRedisList*   redisplList )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCRebreakCol" ) );

    try {
        col->Rebreak2( redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCRebreakCol" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCFS_SetRecompose( scColumn* col, Bool tf )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCFS_SetRecompose" ) );

    try {
        if ( col )
            col->SetRecomposition( tf );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCFS_SetRecompose" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCFS_GetRecompose( scColumn*      col,
                                           Bool&          tf )
{
    status stat = scSuccess;

```

```

    EnterMonitor( scString( "SCFS_GetRecompose" ) );

    try {
        tf = col->GetRecomposition();
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCFS_GetRecompose" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCFS_Recompose( scColumn*      col,
                                       scRedisplist*   redisplist )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCFS_Recompose" ) );

    try {
        col->RecomposeFlowset( LONG_MAX, redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCFS_Recompose" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCFS_Recompose( scColumn*      col,
                                       long             ticks,
                                       scRedisplist*   redisplist )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCFS_Recompose" ) );

    try {
        col->RecomposeFlowset( ticks, redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCFS_Recompose" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_New( APPColumn    appName,
                                   scColumn* &   col,
                                   MicroPoint    width,
                                   MicroPoint    depth )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_New" ) );

    try {
        raise_if( width < 0 || depth < 0, scERRinput );
        col = SCNEW scColumn( appName, width, depth );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_New" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSCR_Write( scScrapPtr    scrap,
                                     APPCtxPtr     ctxPtr,
                                     6

```

```

        IOFuncPtr    writeFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSCR_Write" ) );

    try {
        if ( scrap->IsClass( "scColumn" ) ) {
            scColumn* col = (scColumn*)scrap;
            col->ZeroEnumeration();
            col->StartWrite( ctxPtr, writeFunc );
        }
        else if ( scrap->IsClass( "scStream" ) ) {
            scStream* stream = (scStream*)scrap;
            stream->STRZeroEnumeration();
            stream->STRWriteToFile( ctxPtr, writeFunc );
        }
        else
            raise( scERRidentification );
        scTBObj::WriteNullObject( ctxPtr, writeFunc );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSCR_Write" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSCR_ReadCol( scScrapPtr&    scrap,
                                       scSet*          enumTable,
                                       APPCtxPtr        ctxPtr,
                                       IOFuncPtr        readFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSCR_ReadCol" ) );

    try {
        scColumn* col;
        col = (scColumn*)scTBObj::StartRead( enumTable, ctxPtr, readFunc );
        scAssert( scTBObj::StartRead( enumTable, ctxPtr, readFunc ) == 0 );
        scrap = col;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSCR_ReadCol" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSCR_ReadStream( scScrapPtr&    scrapH,
                                       scSet*          enumTable,
                                       APPCtxPtr        ctxPtr,
                                       IOFuncPtr        readFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSCR_ReadStream" ) );

    scrapH = 0;

    try {
        scStream* stream = scStream::STRFromFile( enumTable, ctxPtr, readFunc );
        scAssert( scTBObj::StartRead( enumTable, ctxPtr, readFunc ) == 0 );
        scrapH = stream;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSCR_ReadStream" ) );
    return stat;
}

/* ===== */

```

```

status scIMPL_EXPORT    SCSCR_Get( scScrapPtr    scrap,
                                   scTypeSpecList& tsList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSCR_TSList" ) );

    try {
        if ( scrap->IsClass( "scColumn" ) )
            ((scColumn *)scrap)->GetTSList( tsList );
        else if ( scrap->IsClass( "scContUnit" ) )
            ((scStream*)scrap)->GetTSList( tsList );
        else
            stat = scERRidentification;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSCR_TSList" ) );
    return stat;
}

```

```
/* ----- */
```

```

status scIMPL_EXPORT    SCSCR_Free( scScrapPtr scrap )
{
    status stat = scSuccess;
    long    bytesFreed;

    EnterMonitor( scString( "SCSCR_Free" ) );

    try {
        if ( !scrap )
            ;
        else if ( scrap->IsClass( "scColumn" ) )
            ((scColumn*)scrap)->FreeScrap();
        else if ( scrap->IsClass( "scContUnit" ) )
            ((scContUnit*)scrap)->FreeScrap( bytesFreed );
        else
            raise( scERRidentification );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSCR_Free" ) );
    return stat;
}

```

```
/* ----- */
```

```

status scIMPL_EXPORT    SCCOL_Delete( scColumn*    col,
                                      scRedisplList* redisplList )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCCOL_Delete" ) );

    try {
        col->Delete( redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_Delete" ) );
    return stat;
}

```

```
/* ----- */
```

```

status scIMPL_EXPORT    SCSTR_Read( scStream*&    stream,
                                   scSet*          enumTable,
                                   APPCtxPtr       ctxPtr,
                                   IOFuncPtr       readFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_Read" ) );

```

```

    try {
        stream = scStream::STRFromFile( enumTable, ctxPtr, readFunc );
        scAssert( scTBObj::StartRead( enumTable, ctxPtr, readFunc ) == 0 );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_Read" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_Write( scStream*  stream,
                                     APPCtxPtr  ctxPtr,
                                     IOFuncPtr  writeFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_Write" ) );

    try {
        stream->STRWriteToFile( ctxPtr, writeFunc );
        scTBObj::WriteNullObject( ctxPtr, writeFunc );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_Write" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_Cut ( scStream*  streamID,
                                     scRedisplList*  redisplList )
{
    status      stat = scSuccess;
    scColumn*   col;
    scTextline* txl;

    EnterMonitor( scString( "SCSTR_Cut" ) );

    try {
        txl = streamID->GetFirstline();
        if ( txl )
            col = txl->GetColumn( );
        else
            col = scColumn::FindFlowset( streamID );

        if ( col )
            col->FlowsetCutStream ( streamID, redisplList );
        else
            raise( scERRstructure );
    }
    IGNORE_RERAISE;

    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_Copy( const scStream* stream,
                                     scStream*&      newStream )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_Copy" ) );

    try {
        stream->STRCopy( newStream );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_Copy" ) );
    return stat;
}

```

```

}

/* ===== */

status scIMPL_EXPORT SCFS_PasteStream ( scColumn*      col,
                                         scStream*      streamID,
                                         scRedisList*    redisList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCFS_PasteStream" ) );

    try {
        col->FlowsetPasteStream( streamID, redisList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCFS_PasteStream" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT SCSTR_Clear ( scStream*      stream,
                                   scRedisList*    redisList )
{
    status      stat = scSuccess;
    scColumn*   col;
    scTextline* txl;

    EnterMonitor( scString( "SCSTR_Clear" ) );

    try {
        if ( stream ) {
            txl = stream->GetFirstline();
            if ( txl ) {
                col = txl->GetColumn( );
                if ( col )
                    col->FlowsetClearStream ( redisList );
                else
                    raise( scERRstructure );
            }
            else if ( stream->FindColumn( col ) )
                col->FlowsetClearStream( redisList );
            else
                /* if no layout structure associated with stream */
                stream->STRFree();
        }
        else
            raise( scNoAction );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_Clear" ) );
    return stat;
}

/* ===== */
// Extracts a scContUnit from a scStreamLocation for use with SCSTR_Split

status scIMPL_EXPORT SCSEL_GetContUnit( scContUnit*& mark,
                                         scContUnit*& point,
                                         const scSelection* sl )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_GetContUnit" ) );

    try {
        sl->GetContUnits( mark, point );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_GetContUnit" ) );
}

```

```

    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_Split( scStream*    stream1,
                                     scContUnit*   cu,
                                     scStream*&    stream2 )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_Split" ) );

    try {
        stream2 = stream1->Split( cu );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_Split" ) );
    return stat;
}

/* ===== */
/* compare streams for equality, this tests content and specs
 * scSuccess == equality
 */

status scIMPL_EXPORT    SCSTR_Compare( const scStream* str1,
                                       const scStream* str2 )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_Compare" ) );

    try {
        stat = str1->Compare( str2 ) ? scSuccess : scNoAction;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_Compare" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_SetSize( scColumn*    col,
                                       MicroPoint    width,
                                       MicroPoint    depth,
                                       scRedisList*   redisplist )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_SetSize" ) );

    try {
        if ( width < 0 || depth < 0 )
            raise( scERRinput );

        col->Resize( width, depth, redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_SetSize" ) );
    return stat;
}

/* ===== */
// is there any text associated with this column

status scIMPL_EXPORT    SCCOL_HasText( scColumn* col )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_HasText" ) );
    11
}

```

```

    try {
        stat = col->HasText( ) ? scSuccess : scNoAction;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_HasText" ) );
    return stat;
}

/* ===== */

// tests to see if there is more text than is in this column
// this would set the flag to true if:
//     there is text in subsequent linked columns
//     there is unformatted text that will not fit in this column

status scIMPL_EXPORT    SCCOL_MoreText( scColumn*    col,
                                        Bool&         flag )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_MoreText" ) );

    try {
        flag = col->MoreText( );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_MoreText" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_LinePositions ( scColumn*    col,
                                              scLineInfoList* lineInfo,
                                              long&         nLines,
                                              Bool&         moreText )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_LinePositions" ) );

    try {
        col->LineInfo( lineInfo, nLines, moreText );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_LinePositions" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_SetVertFlex ( scColumn*    col,
                                           scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_SetVertFlex" ) );

    try {
        col->SetVertFlex ( true, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_SetVertFlex" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_ClearVertFlex ( scColumn*    col,
                                              scRedisplList* redisplList )
{

```



```

    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_ClearVertFlex" ) );

    try {
        col->SetVertFlex ( false, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_ClearVertFlex" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_SetHorzFlex ( scColumn*    col,
                                           scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_SetHorzFlex" ) );

    try {
        col->SetHorzFlex ( true, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_SetHorzFlex" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_ClearHorzFlex ( scColumn*    col,
                                           scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_ClearHorzFlex" ) );

    try {
        col->SetHorzFlex ( false, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_ClearHorzFlex" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_GetVertFlex( scColumn*    col,
                                           Bool&         tf )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_GetVertFlex" ) );

    try {
        tf = col->GetVertFlex( );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_GetVertFlex" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_GetHorzFlex ( scColumn*    col,
                                           Bool&         tf )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_GetHorzFlex" ) );

    try {
        tf = col->GetHorzFlex( );
    }

```

```

    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_GetHorzFlex" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_GetFlowDirection( scColumn*    col,
                                                scFlowDir&    flodir )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_GetFlowDirection" ) );

    try {
        flodir = col->GetFlowdir();
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_GetFlowDirection" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_SetFlowDirection( scColumn*    col,
                                                const scFlowDir&    flodir )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_SetFlowDirection" ) );

    try {
        col->FlowsetSetFlowdir( flodir );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_SetFlowDirection" ) );
    return stat;
}

/* ===== */

#ifdef scColumnShape
status scIMPL_EXPORT    SCCOL_PastePoly ( scColumn*    col,
                                           const scVertex*    vert,
                                           scRedisplist*    redisplist )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_PastePoly" ) );

    try {
        col->PastePoly ( vert, redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_PastePoly" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_ClearPoly ( scColumn*    col,
                                           scRedisplist*    redisplist )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_ClearPoly" ) );

    try {
        col->ClearShape( redisplist );
    }
    IGNORE_RERAISE;
}

```

```

    ExitMonitor( scString( "SCCOL_ClearPoly" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_CopyPoly ( scColumn* col,
                                         scVertex*& vert )
{
    status stat = scSuccess;
    /*CLIPSTUFF*/
    EnterMonitor( scString( "SCCOL_CopyPoly" ) );

    try {
        col->CopyPoly( vert );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_CopyPoly" ) );
    return stat;
}

/*===== REGIONS =====*/

status scIMPL_EXPORT    SCCOL_PasteRgn ( scColumn* col,
                                         const HRgnHandle rgnH,
                                         scRedisplist* redisplist )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_PasteRgn" ) );

    try {
        col->PasteRgn( rgnH, redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_PasteRgn" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_ClearRgn ( scColumn* col,
                                         scRedisplist *redisplist )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_ClearRgn" ) );

    try {
        col->ClearShape( redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_ClearRgn" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_CopyRgn ( scColumn* col,
                                         HRgnHandle& rgnH )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_CopyRgn" ) );

    try {
        col->CopyRgn( rgnH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_CopyRgn" ) );
    return stat;
}

```

```
/* ===== */
```

```
status scIMPL_EXPORT    SCHRG_New( HRgnHandle& hrgH,
                                   MicroPoint sliverSize )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_New" ) );

    try {
        hrgH = NewHRgn( sliverSize );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_New" ) );
    return stat;
}
```

```
/* ===== */
```

```
status scIMPL_EXPORT    SCHRG_Dispose( HRgnHandle hrgH )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_Dispose" ) );

    try {
        DisposeHRgn( hrgH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_Dispose" ) );
    return stat;
}
```

```
/* ===== */
```

```
status scIMPL_EXPORT    SCHRG_Empty( HRgnHandle hrgH )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_Empty" ) );

    try {
        stat = EmptyHRgn( hrgH ) ? scSuccess : scNoAction;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_Empty" ) );
    return stat;
}
```

```
/* ===== */
```

```
status scIMPL_EXPORT    SCHRG_Equal( const HRgnHandle a,
                                   const HRgnHandle b )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_Equal" ) );

    try {
        stat = EqualHRgn( a, b ) ? scSuccess : scNoAction;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_Equal" ) );
    return stat;
}
```

```
/* ===== */
```

```

status scIMPL_EXPORT  SCHRGnPtIn( const HRgnHandle  hrgH,
                                const scMuPoint&   pt )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRGnPtIn" ) );

    try {
        stat = PtInHRgn( hrgH, pt ) ? scSuccess : scNoAction;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRGnPtIn" ) );
    return stat;
}

```

```
/* ----- */
```

```

status scIMPL_EXPORT  SCHRGN_Rect( HRgnHandle  hrgH,
                                const scXRect& xrect )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRGN_Rect" ) );

    try {
        RectHRgn( hrgH, xrect );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRGN_Rect" ) );
    return stat;
}

```

```
/* ----- */
```

```

status scIMPL_EXPORT  SCHRGN_Poly( HRgnHandle  hrgH,
                                const scVertex* verts )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRGN_Poly" ) );

    try {
        PolyHRgn( hrgH, verts );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRGN_Poly" ) );
    return stat;
}

```

```
/* ----- */
```

```

status scIMPL_EXPORT  SCHRGN_Copy( HRgnHandle  dstRgn,
                                const HRgnHandle srcRgn )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRGN_Copy" ) );

    try {
        CopyHRgn( dstRgn, srcRgn );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRGN_Copy" ) );
    return stat;
}

```

```
/* ----- */
```

```

status scIMPL_EXPORT  SCHRGN_Sect( const HRgnHandle  a,
                                17

```

```

                                const HRgnHandle  b,
                                HRgnHandle        dstRgnH )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_Sect" ) );

    try {
        SectHRgn( a, b, dstRgnH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_Sect" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCHRG_N_Union( const HRgnHandle  a,
                                       const HRgnHandle  b,
                                       HRgnHandle        dstRgnH )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_Union" ) );

    try {
        UnionHRgn( a, b, dstRgnH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_Union" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCHRG_N_Diff( const HRgnHandle  a,
                                       const HRgnHandle  b,
                                       HRgnHandle        dstRgnH )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_Diff" ) );

    try {
        DiffHRgn( a, b, dstRgnH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_Diff" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCHRG_N_Xor( const HRgnHandle  a,
                                       const HRgnHandle  b,
                                       HRgnHandle        dstRgnH )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_Xor" ) );

    try {
        XorHRgn( a, b, dstRgnH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_Xor" ) );
    return stat;
}

```

```

/* ----- */

status scIMPL_EXPORT    SCHRG_N_Translate( HRgnHandle    hrgH,
                                           MicroPoint    x,
                                           MicroPoint    y )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_Translate" ) );

    try {
        TranslateHRgn( hrgH, x, y );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_Translate" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCHRG_N_Inset( HRgnHandle    hrgH,
                                       MicroPoint    x,
                                       MicroPoint    y )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_Inset" ) );

    try {
        InsetHRgn( hrgH, x, y, true );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_Inset" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCHRG_N_SetEmpty( HRgnHandle hrgH )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_SetEmpty" ) );

    try {
        SetEmptyHRgn( hrgH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_SetEmpty" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCHRG_N_SliverSize( HRgnHandle    hrgH,
                                              MicroPoint&    sliverSize )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_SliverSize" ) );

    try {
        sliverSize = RGNSliverSize( hrgH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_SliverSize" ) );
    return stat;
}

```

```

/* ----- */

status scIMPL_EXPORT    SCHRG_N_RectIn( const HRgnHandle hrgH,
                                         const scXRect&   xrect )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCHRG_N_RectIn" ) );

    try {
        stat = RectInHRgn( hrgH, xrect ) ? scSuccess : scNoAction;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCHRG_N_RectIn" ) );
    return stat;
}

/* ----- */

#endif

/* ----- */

status scIMPL_EXPORT    SCCOL_Update( scColumn*      col,
                                     const scXRect&   xrect,
                                     APPDrwCtx        mat )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_Update" ) );

    try {
        col->Draw ( xrect, mat );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_Update" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCCOL_UpdateLine( scColumn*      col,
                                          scImmediateRedisp& lineDamage,
                                          APPDrwCtx        mat )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_UpdateLine" ) );

    try {
        col->UpdateLine( lineDamage, mat );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_UpdateLine" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCCOL_TSList ( scColumn*      col,
                                       scTypeSpecList& tsList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_TSList" ) );

    try {
        col->GetTSList( tsList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_TSList" ) );
}

```



```

    return stat;
}

/* ===== */
status scIMPL_EXPORT SCSTR_TSList ( scStream*      stream,
                                   scTypeSpecList& tsList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_TSList" ) );

    try {
        stream->STRGetTSList( tsList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_TSList" ) );
    return stat;
}

/* ===== */
status scIMPL_EXPORT SCSTR_ParaTSList ( scStream*      stream,
                                       scTypeSpecList& tsList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_ParaTSList" ) );

    try {
        stream->GetParaTSList( tsList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_ParaTSList" ) );
    return stat;
}

/* ===== */
status scIMPL_EXPORT SCSTR_CHTSList ( scStream*      stream,
                                     scSpecLocList& csList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_CHTSList" ) );

    try {
        stream->GetCharSpecList( csList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_CHTSList" ) );
    return stat;
}

/* ===== */
status scIMPL_EXPORT SCCOL_SetDepthNVJ( scColumn*      col,
                                       MicroPoint      depth,
                                       scRedisplList    *redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_SetDepthNVJ" ) );

    try {
        col->SetDepthNVJ ( depth, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_SetDepthNVJ" ) );
    return stat;
}

/* ===== */

```

```

status scIMPL_EXPORT SCCOL_FlowJustify( scColumn* col,
                                         eVertJust attributes )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_FlowJustify" ) );

    try {
        col->SetVJ( attributes );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_FlowJustify" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT SCSTR_ChCount( scStream* stream,
                                     long& count )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_ChCount" ) );

    try {
        stream->ChCount( count );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_ChCount" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT SCSEL_TSList ( scSelection* selection,
                                     scTypeSpecList& tsList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_TSList" ) );

    try {
        selection->GetTSList( tsList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_TSList" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT SCSEL_CHTSList( scSelection* selection,
                                     scSpecLocList& csList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_CHTSList" ) );

    try {
        selection->GetCharSpecList( csList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_CHTSList" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT SCSEL_PARATSList( scSelection* sel,
                                       scSpecLocList& cslist )
{
    status stat = scSuccess;

```

```

    EnterMonitor( scString( "SCSEL_PARATSList" ) );

    try {
        sel->GetParaSpecList( cslist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_PARATSList" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_PARATSList( scSelection*    sel,
                                         scTypeSpecList&  tsList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_PARATSList" ) );

    try {
        sel->GetParaSpecList( tsList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_PARATSList" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_PARATSList( scStream*      stream,
                                         scSpecLocList&   cslist )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_PARATSList" ) );

    try {
        stream->GetParaSpecList( cslist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_PARATSList" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_CHTSListSet ( scStream*      str,
                                         const scSpecLocList& csList,
                                         scRedisplList*    redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_CHTSListSet" ) );

    try {
        scAssert( str == csList.GetStream() );
        str->SetCharSpecList( csList, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_CHTSListSet" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_PARATSListSet( scStream*      str,
                                         const scSpecLocList& cslist,
                                         scRedisplList*    rInfo )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_PARATSListSet" ) );

```

```

    try {
        scAssert( str == cslist->GetStream() );
        str->SetParaSpecList( cslist, rInfo );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_PARATSListSet" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCEExternalSize ( scColumn*   col,
                                           long&       size )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCEExternalSize" ) );

    try {
        col->ExternalSize( size );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCEExternalSize" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCTB_ZeroEnumeration( void )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCTB_ZeroEnumeration" ) );

    try {
        scColumn*   col = scColumn::GetBaseContextList();
        for ( ; col; col = col->GetContext() )
            col->ZeroEnumeration();
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCTB_ZeroEnumeration" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSET_InitRead( scSet*& enumTable,
                                         long    maxsize )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSET_InitRead" ) );

    try {
        enumTable = SCNEW scSet;
        enumTable->SetNumSlots( maxsize );
        enumTable->SetRetainMem( true );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSET_InitRead" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSET_FiniRead( scSet*       enumTable,
                                       scRedisplList* redisplList )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCSET_FiniRead" ) );

```

```

    try {
        delete enumTable, enumTable = 0;
        scColumn::Update( redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSET_FiniRead" ) );

    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSET_Abort( scSet*& enumTable )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSET_Abortt" ) );
    try {
        enumTable->DeleteAll();
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSET_Abortt" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCOBJ_PtrRestore( scTBObj*   obj,
                                           scSet*    enumTable )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCOBJ_PtrRestore" ) );
    try {
        long    i,
                limit = enumTable->GetNumItems();

        for ( i = 0; i < limit; i++ ) {
            scTBObj* ptr = (scTBObj*)enumTable->Get( i );
            if ( ptr )
                ptr->RestorePointers( enumTable );
        }
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCOBJ_PtrRestore" ) );

    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_QueryInkExtents( scColumn*   col,
                                                scXRect& xrect )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCCOL_QueryInkExtents" ) );

    try {
        col->ComputeInkExtents();
        xrect = col->GetInkExtents();
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_QueryInkExtents" ) );
    return stat;
}

/* ===== */

```

```

status scIMPL_EXPORT SCCOL_QueryMargins( scColumn* col,
                                         scXRect& xrect )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCCOL_QueryMargins" ) );

    try {
        col->QueryMargins( xrect );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_QueryMargins" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT SCCOL_Size( scColumn* col,
                                scSize& size )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_Size" ) );

    try {
        col->QuerySize( size );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_Size" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT SCOBJ_Enumerate( scTBObj* obj,
                                     long& objEnumerate )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCOBJ_Enumerate" ) );

    try {
        if ( obj->IsClass( "scColumn" ) )
            ((scColumn*)obj)->Enumerate( objEnumerate );
        else if ( obj->IsClass( "scContUnit" ) )
            ((scStream*)obj)->DeepEnumerate( objEnumerate );
        else
            raise( scERRstructure );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCOBJ_Enumerate" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT SCCOL_Link( scColumn* col1,
                                scColumn* col2,
                                scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_Link" ) );

    try {
        col1->Link( col2, true, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_Link" ) );
    return stat;
}

```

```
/* ===== */
```

```
status scIMPL_EXPORT    SCCOL_Unlink ( scColumn*    col,
                                     scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_Unlink" ) );

    try {
        col->Unlink( redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_Unlink" ) );
    return stat;
}
```

```
/* ===== */
```

```
status scIMPL_EXPORT    SCFS_Split( scColumn*    col1,
                                    scColumn*    col2 )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCFS_Split" ) );

    try {
        col1->BreakChain( col2 );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCFS_Split" ) );
    return stat;
}
```

```
/* ===== */
```

```
status scIMPL_EXPORT    SCCOL_GetStream ( scColumn*    col,
                                           scStream*&    stream )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_GetStream" ) );

    try {
        stream = col->GetStream ();
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_GetStream" ) );
    return stat;
}
```

```
/* ===== */
```

```
status scIMPL_EXPORT    SCFS_ReadTextFile ( scColumn*    col,
                                           TypeSpec      spec,
                                           APPCtxPtr      ctxPtr,
                                           IOFuncPtr      readFunc,
                                           scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCFS_ReadTextFile" ) );

    try {
        col->ReadTextFile( spec, ctxPtr, readFunc, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCFS_ReadTextFile" ) );
    return stat;
}
```

```
/* ----- */
```

```
status scIMPL_EXPORT    SCWriteTextFile ( scStream* stream,
                                           APPCtxPtr ctxPtr,
                                           IOFuncPtr writeFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCWriteTextFile" ) );

    try {
        stream->STRWriteTextFile( ctxPtr, writeFunc, false );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCWriteTextFile" ) );
    return stat;
}
```

```
/* ----- */
```

```
status scIMPL_EXPORT    SCTextFileToScrap ( scScrapPtr& scrapH,
                                           APPCtxPtr  ctxPtr,
                                           IOFuncPtr  readFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCTextFileToScrap" ) );

    try {
        TypeSpec nullSpec;

        scStream* stream = scStream::ReadTextFile( nullSpec, ctxPtr, readFunc, 0 );
        scrapH = stream;
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCTextFileToScrap" ) );
    return stat;
}
```

```
/* ----- */
```

```
status scIMPL_EXPORT    SCAPPTXT_Alloc( stTextImportExport*& apptext )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCAPPTXT_Alloc" ) );

    apptext = 0;

    try {
        apptext = &stTextImportExport::MakeTextImportExport( 1 );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCAPPTXT_Alloc" ) );

    return stat;
}
```

```
/* ----- */
```

```
status scIMPL_EXPORT    SCAPPTXT_Delete( stTextImportExport* apptext )
{
    status stat = scSuccess;

    EnterMonitor( scString( "SCAPPTXT_Delete" ) );

    try {
        apptext->release();
    }
    IGNORE_RERAISE;
}
```



```

    ExitMonitor( scString( "SCSTR_TXT_Delete" ) );
    return stat;
}

/* ===== */
status scIMPL_EXPORT    SCFS_PasteAPPText( scColumn*      col,
                                           stTextImportExport&  appText,
                                           scRedisplList*  redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCFS_PasteAPPText" ) );

    try {
        col->PasteAPPText( appText, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCFS_PasteAPPText" ) );
    return stat;
}

/* ===== */
status scIMPL_EXPORT    SCSEL_PasteAPPText( scSelection*   sel,
                                           stTextImportExport&  appText,
                                           scRedisplList*  redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_PasteAPPText" ) );

    try {
        sel->PasteAPPText( appText, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_PasteAPPText" ) );
    return stat;
}

/* ===== */
status scIMPL_EXPORT    SCSTR_GetAPPText( scStream*      stream,
                                           stTextImportExport&  appText )
{
    status stat = scSuccess;
    /*CLIPSTUFF*/
    EnterMonitor( scString( "SCSTR_GetAPPText" ) );

    try {
        stream->CopyAPPText ( appText );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_GetAPPText" ) );
    return stat;
}

/* ===== */
status scIMPL_EXPORT    SCSEL_GetAPPText( scSelection*   selection,
                                           stTextImportExport&  appText )
{
    status stat = scSuccess;
    /*CLIPSTUFF*/
    EnterMonitor( scString( "SCSEL_GetAPPText" ) );

    try {
        selection->CopyAPPText ( appText );
    }
    IGNORE_RERAISE;
}

```

```

    ExitMonitor( scString( "SCCOL_GetAPPTText" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCCOL_Write( scColumn*   col,
                                     APPCtxPtr   ctxPtr,
                                     IOFuncPtr   writeFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_Write" ) );

    try {
        col->StartWrite( ctxPtr, writeFunc );
        scTBObj::WriteNullObject( ctxPtr, writeFunc );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_Write" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCCOL_Read( APPColumn      appName,
                                     scColumn*&     col,
                                     scSet*          enumTable,
                                     APPCtxPtr       ctxPtr,
                                     IOFuncPtr       readFunc )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_Read" ) );

    try {
        col = (scColumn*)scTBObj::StartRead( enumTable, ctxPtr, readFunc );
        scAssert( scTBObj::StartRead( enumTable, ctxPtr, readFunc ) == 0 );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_Read" ) );
    return stat;
}

/* ----- */
/* ----- SELECTION MESSAGES ----- */
/* ----- */

status scIMPL_EXPORT    SCCOL_ClickEvaluate ( scColumn*   col,
                                               const scMuPoint& pt,
                                               REAL&       dist )
{
    status      stat = scSuccess;
    scMuPoint   cmpt = pt;

    EnterMonitor( scString( "SCCOL_ClickEvaluate" ) );

    try {
        col->ClickEvaluate( cmpt, dist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_ClickEvaluate" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCCOL_StartSelect( scColumn*   col,
                                           const scMuPoint& pt,
                                           HiliteFuncPtr DrawRect,
                                           APPDrawCtx   mat,
                                           scSelection*& selectID )

```

```

{
    status      stat = scSuccess;
    EnterMonitor( scString( "SCCOL_StartSelect" ) );

#if SCDEBUG > 1
    SCDebugTrace( 2, scString( "SCCOLStartSelect %d %d\n" ), pt.x, pt.y );
#endif

    try {
        col->StartClick( pt, DrawRect, mat, selectID );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_StartSelect" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_StartSelect( scColumn*      col,
                                           scStreamLocation& mark,
                                           const scMuPoint& pt,
                                           HiliteFuncPtr DrawRect,
                                           APPDrwCtx mat,
                                           scSelection*& selectID )

{
    status      stat = scSuccess;

    EnterMonitor( scString( "SCCOL_StartSelect" ) );

    try {
        col->StartShiftClick( mark, pt, DrawRect, mat, selectID );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_StartSelect" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_ExtendSelect( scColumn*      col,
                                           const scMuPoint& pt,
                                           HiliteFuncPtr DrawRect,
                                           APPDrwCtx,
                                           scSelection* select )

{
    status      stat = scSuccess;

    EnterMonitor( scString( "SCCOL_ExtendSelect" ) );

    // SCDebugTrace( 0, scString( "SCCOLExtendSelect ENTER %d %d\n" ), pt.x, pt.y );

    try {
        raise_if( select == 0, scERRinput );
        col->ContinueClick( pt, DrawRect, select );
    }
    IGNORE_RERAISE;

    // SCDebugTrace( 0, scString( "SCCOLExtendSelect EXIT %d %d\n" ), pt.x, pt.y );

    ExitMonitor( scString( "SCCOL_ExtendSelect" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_InitialSelect ( scColumn*      col,
                                              TypeSpec&      typespec,
                                              scSelection*& selectID )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_InitialSelect" ) );
    31
}

```

```

    try {
        col->InitialSelection( typespec, selectID );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_InitialSelect" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_Decompose ( scSelection*    select,
                                           scStreamLocation& mark,
                                           scStreamLocation& point )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_Decompose" ) );

    try {
        select->Decompose( mark, point );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_Decompose" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_Decompose2( scSelection*    select,
                                           scStreamLocation& mark,
                                           scStreamLocation& point )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_Decompose" ) );

    try {
        select->Decompose2( mark, point );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_Decompose" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_Invalidate( scSelection*& select )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_Invalidate" ) );

    try {
        if ( select )
            select->Invalidate();
    }
    IGNORE_RERAISE;

    select = 0;

    ExitMonitor( scString( "SCSEL_Invalidate" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_Restore( const scStream*    stream,
                                       const scStreamLocation& mark,
                                       const scStreamLocation& point,
                                       scSelection*&      select,
                                       Bool                geometryChange )
{
    status stat = scSuccess;

```

```

EnterMonitor( scString( "SCCOL_Restore" ) );

try {
    scColumn* col = scColumn::FindFlowset( stream );

    // we cannot create a selection if there is no layout
    raise_if( col == 0, scERRstructure );

    select = col->FlowsetGetSelection();

    select->Restore( &mark, &point, stream, geometryChange );
}
IGNORE_RERAISE;

ExitMonitor( scString( "SCSEL_Restore" ) );
return stat;
}

/* ===== */

status scIMPL_EXPORT    SCCOL_SelectSpecial( scColumn*      col,
                                             const scMuPoint& pt,
                                             eSelectModifier selectMod,
                                             scSelection*&  selectID )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCCOL_SelectSpecial" ) );

    try {
        col->SelectSpecial( pt, selectMod, selectID );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCCOL_SelectSpecial" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_Move( scSelection*  select,
                                     eSelectMove   moveSelect )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_Move" ) );

    try {
        select->MoveSelect( moveSelect );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_Move" ) );

    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_Extend( scSelection*  select,
                                       eSelectMove   moveSelect )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_Move" ) );

    try {
        select->Extend( moveSelect );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_Move" ) );
}

```

```

    return stat;
}

/* ===== */
status scIMPL_EXPORT SCSEL_Hilite( scSelection* select,
                                   HiliteFuncPtr DrawRect )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_Hilite" ) );

    try {
        select->ValidateHilite( DrawRect );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_Hilite" ) );
    return stat;
}

/* ===== */
/*===== EDITING MESSAGES =====*/
status scIMPL_EXPORT SCSEL_InsertKeyRecords( scSelection* select,
                                              short keyCount,
                                              scKeyRecord* keyRecords, /* array of key recds */
                                              scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_InsertKeyRecords" ) );

    try {
        select->KeyArray( keyCount, keyRecords, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_InsertKeyRecords" ) );
    return stat;
}

/* ===== */
status SCSEL_InsertField( scSelection* sel,
                          const clField& field,
                          TypeSpec& spec,
                          scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_InsertAnnotation" ) );

    try {
        sel->InsertField( field, spec, redisplList );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_InsertAnnotation" ) );
    return stat;
}

/* ===== */
status scIMPL_EXPORT SCSEL_SetTextStyle ( scSelection* selection,
                                           TypeSpec style,
                                           scRedisplList* redisplList )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_SetTextStyle" ) );

    try {
        selection->SetStyle( style, redisplList );
    }
    IGNORE_RERAISE;
}

```

```

    ExitMonitor( scString( "SCSEL_SetTextStyle" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCSEL_TextTrans ( scSelection*  select,
                                          eChTranType   trans,
                                          int            numChars,      // this is a modifier for th
                                          scRedisplist *redisplist )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_TextTrans" ) );

    try {
        select->TextTrans( trans, numChars, redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_TextTrans" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCSEL_CutText ( scSelection*  selection,
                                         scScrapPtr&   scrap,
                                         scRedisplist* redisplist )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_CutText" ) );

    try {
        selection->CutText( scrap, redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_CutText" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCSEL_ClearText( scSelection*  selection,
                                         scRedisplist* redisplist )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_ClearText" ) );

    try {
        selection->ClearText( redisplist, true );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_ClearText" ) );
    return stat;
}

/* ----- */

status scIMPL_EXPORT    SCSEL_CopyText ( scSelection*  selection,
                                         scScrapPtr&   scrap )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_CopyText" ) );

    try {
        selection->CopyText( scrap );
    }
    IGNORE_RERAISE;
}

```

```

    ExitMonitor( scString( "SCSEL_CopyText" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_PasteText ( scSelection*  selection,
                                           scScrapPtr    scrap,
                                           TypeSpec      style,
                                           scRedisplist* redisplist )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_PasteText" ) );

    try {
        selection->PasteText( (scStream*)scrap, style, redisplist );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_PasteText" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSCR_ConToSys ( scScrapPtr      scrap,
                                           SystemMemoryObject& pSysConBlock )
{
    status stat = scSuccess;
    scContUnit* para = (scContUnit*)scrap;
    EnterMonitor( scString( "SCSCR_ConToSys" ) );

    try {
        if ( scrap->IsClass( "scColumn" ) )
            para = ((scColumn*)scrap)->GetStream();

        if ( para->IsClass( "scContUnit" ) )
            ((scStream*)para)->STRWriteMemText( false, pSysConBlock );
        else
            raise( scERRidentification );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSCR_ConToSys" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSCR_SysToCon( scScrapPtr&      scrapP,
                                           const scChar*  sysScrapPtr,
                                           TypeSpec        ts )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSCR_SysToCon" ) );

    try {
        scrapP = scStream::ReadMemText( ts, sysScrapPtr );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSCR_SysToCon" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_Iter( scSelection*  select,
                                     SubstituteFunc func,
                                     36

```



```

        scRedisplist*  damage )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_Iter" ) );

    try {
        select->Iter( func, damage );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_Iter" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_Iter( scStream*      stream,
                                   SubstituteFunc  func,
                                   scRedisplist*   damage )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_Iter" ) );

    try {
        stream->Iter( func, damage );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_Iter" ) );
    return stat;
}

/* ===== */
// deprecated
status scIMPL_EXPORT    SCSTR_Search( scStream*      stream,
                                       const UCS2*    findString,
                                       SubstituteFunc  func,
                                       scRedisplist*   damage )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_Search" ) );

    try {
        throw( scERRnotImplemented );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_Search" ) );
    return scSuccess;
}

/* ===== */
// deprecated
status scIMPL_EXPORT    SCSEL_FindString( scSelection* select,
                                           const UCS2*   findString )
{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_FindString" ) );

    try {
        throw( scERRnotImplemented );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_FindString" ) );
    return scSuccess;
}

/* ===== */

```

```

status scIMPL_EXPORT    SCSEL_GetStream( const scSelection* selection,
                                         scStream*&          stream,
                                         TypeSpec&          ts )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_GetStream" ) );

    try {
        stream = selection->GetStream();
        ts     = selection->GetSpecAtStart();
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_GetStream" ) );
    return scSuccess;
}

/* ===== */

status scIMPL_EXPORT    SCSTR_NthParaSelect( scStream*      streamID,
                                             long            nthPara,
                                             scSelection*    select )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSTR_NthParaSelect" ) );

    try {
        select->NthPara( streamID, nthPara );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSTR_NthParaSelect" ) );
    return stat;
}

/* ===== */

#ifdef _RUBI_SUPPORT
status scIMPL_EXPORT    SCSEL_GetAnnotation( scSelection*    select,
                                             int              nth,
                                             scAnnotation&    annotation )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_GetAnnotation" ) );

    try {
        select->GetAnnotation( nth, annotation );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_GetAnnotation" ) );
    return stat;
}

/* ===== */

status scIMPL_EXPORT    SCSEL_ApplyAnnotation( scSelection*    select,
                                              const scAnnotation& annotation,
                                              scRedisplList*    redisplayListH )

{
    status stat = scSuccess;
    EnterMonitor( scString( "SCSEL_ApplyAnnotation" ) );

    try {
        select->ApplyAnnotation( annotation, redisplayListH );
    }
    IGNORE_RERAISE;

    ExitMonitor( scString( "SCSEL_ApplyAnnotation" ) );
}

```

```

    return stat;
}

#endif

/* ===== */
/* ===== */
/* ===== */

void scIMPL_EXPORT      SCCOL_InvertExtents( scColumn*      col,
                                             HiliteFuncPtr  func,
                                             APPDrwCtx      drawCtx )
{
    status stat = scSuccess;
    try {
        col->InvertExtents( func, drawCtx );
    }
    IGNORE_RERAISE;
}

/* ===== */

#if SCDEBUG > 1

long scIMPL_EXPORT      SCCOL_DebugSize( scColumn* col )
{
    return sizeof( scColumn ) + ( col->GetLinecount( ) * sizeof( scTextline ) );
}

/* ===== */

long scIMPL_EXPORT      SCSTR_DebugSize( scStream* stream )
{
    return stream->STRDebugSize( );
}

/* ===== */

void scIMPL_EXPORT      SCCOL_InvertRegion( scColumn*      col,
                                             HiliteFuncPtr  func,
                                             APPDrwCtx      drawCtx )
{
    status stat = scSuccess;
    try {
        scFlowDir fd( col->GetFlowdir( ) );
        if ( col->GetRgn( ) )
            RGNInvertSlivers( col->GetRgn(), drawCtx, func, col->GetSize(), fd.IsVertical() );
    }
    IGNORE_RERAISE;
}

/* ===== */

void scIMPL_EXPORT      SCDebugColumnList( void )
{
    scColumn* col;

    SCDebugTrace( 0, scString( "Toolbox Column list start\n" ) );

    for ( col = scColumn::GetBaseContextList( ); col; col = col->GetContext( ) ) {
        SCDebugTrace( 0, scString( "\tcol 0x%08x appname 0x%08x\n" ), col, col->GetAPPName() );
    }
    SCDebugTrace( 0, scString( "Toolbox Column list end\n" ) );
}

/* ===== */

void scIMPL_EXPORT      SCDebugColumn( scColumn*      col,
                                       int              contentLevel )
{
    SCDebugTrace( 0, scString( "Column 0x%08x appname 0x%08x %d %d %s\n" ),

```

```

        col->GetPName(),
        col->Width(), col->Depth(),
        col->GetVertFlex() ? "VFLEX" : "noflex" );

    if ( contentLevel ) {
        scContUnit* p;

        for ( p = col->GetStream(); p; p = p->GetNext() )
            SCDebugTrace( 0, scString( "\\tpara 0x%08x\\n" ), p );
    }
}

/* ===== */

void scIMPL_EXPORT SCDebugParaSpecs( scSelection* sel )
{
    #if 1
        scSelection sorted( *sel );

        sorted.Sort();

        scContUnit* cu = sorted.GetMark().fPara;
        for ( ; cu && cu != sorted.GetPoint().fPara->GetNext(); cu = cu->GetNext() )
            cu->DebugParaSpecs();
    #else
        if ( sel->IsSliverCursor() )
            sel->GetMark().fPara->DebugParaSpecs();
    #endif
}

/* ===== */

void scIMPL_EXPORT SCSTR_Debug( scStream* str )
{
    str->STRDbgPrintInfo( );
}

/* ===== */

#endif /* DEBUG */

/* ===== */
/* ===== */
/* ===== */

```

File: SCCHAREX.H

\$Header: /Projects/Toolbox/ct/SCCHAREX.H 2 5/30/97 8:45a Wmanis \$

Contains: character exchange from toolbox to outside world

Written by: Lucas

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

#define scIndentSpace 0x0007 deprecated 3/18/96 wam

```

*****/

#ifndef _H_SCCHAREX
#define _H_SCCHAREX

#include "sctypes.h"

#define scLeftArrow ((UCS2)1)
#define scRightArrow ((UCS2)2)
#define scUpArrow ((UCS2)3)
#define scDownArrow ((UCS2)4)
#define scParaSplit ((UCS2)5)
#define scBackSpace ((UCS2)6) // delete character backward
#define scForwardDelete ((UCS2)7) // delete character forward

// the following are characters actually stored in the stream and do have
// real character codes
#define scEndStream 0x0000
/* 0x0001 is taken */
/* 0x0002 is taken */
/* 0x0003 is taken */
/* 0x0004 is taken */
/* 0x0005 is taken */
/* 0x0006 is taken */
#define scEmptySpace 0x0008 /* a horizontal move that is meaningless to the user */
#define scTabSpace 0x0009 /* part of the mac character set */
#define scHardReturn 0x000a /* part of the mac character set */
#define scVertTab 0x000b
#define scField 0x000c /* field character */

/* 0x000d is not taken */
/* 0x000e is not taken */
#define scRulePH 0x000f
/* 0x0010 is not taken */
#define scParaStart 0x0011 /* this has no meaning outside of a report to the client of the
cursor position */
#define scParaEnd 0x0012 /* para break */
#define scQuadCenter 0x0013
#define scQuadLeft 0x0014
#define scQuadRight 0x0015
#define scQuadJustify 0x0016
#define scFixAbsSpace 0x0017 /* absolute fixed space */
#define scFixRelSpace 0x0018 /* relative fixed space stored in rlu's */
#define scFillSpace 0x0019
#define scNoBreakHyph 0x001a
#define scDischHyphen 0x001b
#define scFigureSpace 0x001c
#define scThinSpace 0x001d
#define scEnSpace 0x001e
#define scEmSpace 0x001f

```

```

#define scWordSpace      0x0020
#define scRomanWordSpace scWordSpace // ' ' or 0x20 or 32
#define scKanjiWordSpace 0x8140

#define scBreakingHyphen '-'
#define scNoBreakSpace   0x00a0 /* part of the mac character set */
#define scEnDash         0x00d0
#define scEmDash         0x00d1

inline Bool IsBreakingCharacter( UCS2 ch )
{ return ch == scBreakingHyphen || ch == scEnDash || ch == scEmDash; }

UCS2    CMinputMap( ushort ); /* from APP to Stonehand -
                               * on file importing
                               */
UCS2    CMctToAPP( UCS2 ); /* from Stonehand to APP */
UCS2    CMappToCT( UCS2 ); /* from APP to Stonehand */
int      CMcontent( UCS2 ); /* is keystroke a selection change
                               * or a real input of content
                               */

void      CMmakeKeyRecordTwo( scKeyRecord&,
                              UCS2,
                              GlyphSize,
                              TypeSpec,
                              Bool,
                              scStreamLocation& );

#endif /* _H_SCCHAREX */

```

File: SCCHAR.H

\$Header: /Projects/Toolbox/ct/SCCHFLAG.H 2 5/30/97 8:45a Wmanis \$

Contains: Flags for the glyph processing.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

```

*****/

#ifndef _H_SCCHAR
#define _H_SCCHAR

#define SIZE_OF_MACHINE 256

/* character definitions */
#define MIN_CHARACTER START_STREAM
#define MAX_CHARACTER (SIZE_OF_MACHINE - 1)

struct scCharFlags1 {
    unsigned fFauxChar : 16;           // for alignment purposes
    unsigned fDiscHyph : 1;
    unsigned fNoBreak : 1;
    unsigned fHyphLevel : 3;
    unsigned fAutoKern : 1;
    unsigned fDropCap : 1;             // why do i need this
    unsigned fLineBreak : 1;          // why do i need this
    unsigned fField : 8;
};

struct scCharFlags2 {
    unsigned fauxchar_ : 16;           // for alignment purposes
    unsigned dischyph_ : 1;
    unsigned nobreak_ : 1;
    unsigned hyphlevel_ : 3;
    unsigned autokern_ : 1;
    unsigned dropcap_ : 1;             // why do i need this
    unsigned linebreak_ : 1;          // why do i need this
    unsigned spacepos_ : 2;           // position of space leading or trailing or none in escapeme
    unsigned warichu_ : 2;            // if non-zero represent # lines
    unsigned rubi_ : 1;               // annotated character(s)
    unsigned renmoji_ : 3;            // max target of 7 characters
};

class scCharFlags {
    friend class CharRecord;

public:
    void ClrCJKVVarious( void )
    {
        ClrVarious();
        f2_.spacepos_ = 0;
    }
};

```

```

void      ClrVarious( void )
{
    fl_.fLineBreak = 0;
    fl_.fHyphLevel = 0;
}

int      operator==( const scCharFlags& flags ) const
{
    return f__ == flags.f__;
}

// scCharFlags&  operator=( const scCharFlags& flags )
// {
//     f__ = flags.f__;
//     return *this;
// }

void      SetLineBreak(void)
{
    fl_.fLineBreak = 1;
}

void      ClrLineBreak(void)
{
    fl_.fLineBreak = 0;
}

Bool      IsLineBreak(void) const
{
    return fl_.fLineBreak;
}

void      SetDropCap( void )
{
    fl_.fDropCap = 1;
}

void      ClrDropCap( void )
{
    fl_.fDropCap = 0;
}

Bool      IsDropCap( void ) const
{
    return fl_.fDropCap;
}

void      SetKernBits( void )
{
    fl_.fAutoKern = 1;
}

void      ClrKernBits( void )
{
    fl_.fAutoKern = 0;
}

Bool      IsKernPresent( void ) const
{
    return fl_.fAutoKern;
}

void      SetAutoHyphen( unsigned val )
{
    fl_.fHyphLevel = val;
}

void      ClrAutoHyphen( void )
{
    fl_.fHyphLevel = 0;
}

unsigned  GetHyphLevel( void ) const
{
    return fl_.fHyphLevel;
}

void      SetDiscHyphen( void )

```



```

    {
        f1_.fDiscHyph = 1;
    }
void    ClrDiscHyphen( void )
    {
        f1_.fDiscHyph = 0;
    }
Bool    IsDiscHyphen( void ) const
    {
        return f1_.fDiscHyph;
    }

void    SetNoBreak( void )
    {
        f1_.fNoBreak = 1;
    }
void    ClrNoBreak( void )
    {
        f1_.fNoBreak = 0;
    }
Bool    IsNoBreak( void ) const
    {
        return f1_.fNoBreak;
    }

Bool    IsHyphPresent( void ) const
    {
        return GetHyphLevel()||IsDiscHyphen();
    }

void    ClrAutoBits( void )
    {
        ClrAutoHyphen();
        ClrKernBits();
    }

void    SetRubi( void )
    {
        f2_.rubi_ = 1;
    }
void    ClrRubi( void )
    {
        f2_.rubi_ = 0;
    }
Bool    IsRubi( void ) const
    {
        return f2_.rubi_;
    }

void    SetRenMoji( unsigned val )
    {
        f2_.renmoji_ = val;
    }
void    ClrRenMoji( void )
    {
        f2_.renmoji_ = 0;
    }
unsigned GetRenMoji( void ) const
    {
        return f2_.renmoji_;
    }

void    SetWarichu( unsigned val )
    {
        f2_.warichu_ = val;
    }
void    ClrWarichu( void )
    {
        f2_.warichu_ = 0;
    }
unsigned GetWarichu( void ) const
    {

```

```

        return f2_.warichu_;
    }

    Bool    IsSpecialNihon( void ) const
    {
        return f2_.renmoji_ || f2_.rubi_ || f2_.warichu_;
    }

    void    ClrSpecialNihon( void )
    {
        ClrRubi();
        ClrRenMoji();
        ClrWarichu();
    }

    void    SetSpacePosition( unsigned val )
    {
        f2_.spacepos_ = val;
    }

    void    ClrSpacePosition( void )
    {
        f2_.spacepos_ = 0;
    }

    unsigned GetSpacePosition( void ) const
    {
        return f2_.spacepos_;
    }

    void    SetField( uint8 field )
    {
        f1_.fField = field;
    }

    uint8   GetField( ) const
    {
        return (uint8)f1_.fField;
    }

    Bool    IsBreakable( void ) const
    {
        return !( f2_.renmoji_ || f2_.rubi_ || f2_.warichu_ || f1_.fNoBreak );
    }

private:
    void    ClearMinFlags( void )
    {
        f1_.fDiscHyph = 0;
        f1_.fNoBreak = 0;
        f1_.fHyphLevel = 0;
        f1_.fAutoKern = 0;
        f1_.fDropCap = 0;
        f1_.fLineBreak = 0;
    }

    void    ClearAllFlags( void )
    {
        f2_.dischyph_ = 0;
        f2_.nobreak_ = 0;
        f2_.hyphlevel_ = 0;
        f2_.autokern_ = 0;
        f2_.dropcap_ = 0;
        f2_.linebreak_ = 0;

        f2_.spacepos_ = 0;
        f2_.warichu_ = 0;
        f2_.rubi_ = 0;
        f2_.renmoji_ = 0;
    }

    union {
        scCharFlags1    f1_;
        scCharFlags2    f2_;
        uint32          f__;
    }

```

Page 1

[illegible]

File: SCCOLINF.C

\$Header: /Projects/Toolbox/ct/SCCOLINF.CPP 2 5/30/97 8:45a Wmanis \$

Contains: code to collect column redisplay information

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox Application software is the proprietary
and confidential property of Stonehand Inc.

```

*****/
#include "scpubobj.h"
#include "sccolumn.h"
#include "scglobda.h"
#include "screfdat.h"

/* ----- */
void scRedisplList::ReInit( )
{
}

/* ----- */
scColRedispl* scRedisplList::FindCell( const scColumn* col ) const
{
    long limit = GetNumItems();
    scColRedispl* colredisp = (scColRedispl*)Lock();
    for ( ; limit--; colredisp++ ) {
        if ( colredisp->fColumnID == col ) {
            Unlock();
            return colredisp;
        }
    }
    Unlock();
    return 0;
}

/* ----- */
void scRedisplList::AddCell( scColumn* col )
{
    scAssert( !FindCell( col ) );
    scColRedispl colredisp( col, col->GetAPPName() );
    AppendData( (ElementPtr)&colredisp );
}

/* ----- */
void scRedisplList::AddColumn( const scCOLRefData& colRefData )
{
    Lock();
    scColRedispl* cell = FindCell( colRefData.fCol );
    if ( !cell ) {

```

```

        Unlock();
        AddCell( colRefData.fCol );
        Lock();
        cell = FindCell( colRefData.fCol );
    }

    colRefData.fCol->ComputeInkExtents( );
    cell->fWidth      = colRefData.fCol->Width();
    cell->fDepth      = colRefData.fCol->Depth();
    cell->fExRect      = colRefData.fCol->GetInkExtents();
    cell->fAdditionalText = colRefData.fCol->MoreText( );

    scXRect fRepaintRect( cell->fRepaintRect );
    fRepaintRect.Union( colRefData.fLineDamage );
    cell->fRepaintRect = fRepaintRect;
    cell->fHasRepaint  = fRepaintRect.Valid();

    scXRect fDamageRect( cell->fDamageRect );
    fDamageRect.Union( colRefData.fLineDamage );
    cell->fDamageRect  = fDamageRect;
    cell->fHasDamage    = fDamageRect.Valid();

    Unlock();
}

/* ===== */

void scRedisplist::AddColumn( scColumn* col, scXRect& xrect )
{
    Lock();
    scColRedisplay* cell = FindCell( col );
    if ( !cell ) {
        Unlock();
        AddCell( col );
        Lock();
        cell = FindCell( col );
    }
    col->ComputeInkExtents( );
    cell->fWidth      = col->Width();
    cell->fDepth      = col->Depth();
    cell->fExRect      = col->GetInkExtents();
    cell->fAdditionalText = col->MoreText();
    scXRect fRepaintRect( cell->fRepaintRect );
    fRepaintRect.Union( xrect );
    cell->fRepaintRect = fRepaintRect;
    cell->fHasRepaint  = fRepaintRect.Valid();

    Unlock();
}

/* ===== */

void scRedisplist::SetImmediateRect( scColumn* col,
                                     const scImmediateRedisp& immedredispl )
{
    Lock();

    scColRedisplay* cell = FindCell( col );

    if ( !cell ) {
        Unlock();
        AddCell( col );
        Lock();
        cell = FindCell( col );
    }

    cell->fImmediateRedisplay = true;
    cell->fImmediateArea      = immedredispl;
}

```

```

    Unlock();
}

/* ===== */

status scRedisplist::CL_GetColumnData( APPColumn      appname,
                                       scColRedisplay& data ) const
{
    status      stat      = scSuccess;
    volatile int locked    = false;
    volatile int found     = false;

    try {
        long      limit     = GetNumItems();
        scColRedisplay* colredisp = (scColRedisplay*)Lock();
        locked     = true;

        for ( ; limit--; colredisp++ ) {
            if ( colredisp->fAPPName == appname ) {
                data = *colredisp;
                found = true;
            }
        }
        raise_if( found == false, scERRstructure );
    }
    IGNORE_RERAISE;

    return stat;
}

/* ===== */

```

File: SCCOLUM2.C

\$Header: /Projects/Toolbox/ct/SCCOLUM2.CPP 2 5/30/97 8:45a Wmanis \$

Contains: The code to vj columns and to save the line state
before reformatting and then compare it post reformatting
to determine redisplay.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/
#include "sccolumn.h"
#include "scglobda.h"
#include "sctextli.h"
#include "scpubobj.h"
#include "scmem.h"
#include "scexcept.h"
#include "screfdat.h"
#include "sccallbk.h"
#include "scstcach.h"
#include "scstream.h"
#include <limits.h>

/*****
*****/
struct VJSpace {
    MicroPoint opt;
    MicroPoint max;
    REAL upperBound;
};

/*****
*****/

static void COLFlushTop( scColumn* );
static void COLFlushTopBottom( scColumn* );

enum eDoVJ {
    eVJBottom = 1,
    eVJCenter
};

#define growUnits 30
#define growSize ( sizeof( VJSpace ) * growUnits )

/*****
*****/
/* Add a space record to the end of the handle. */

static void InsertSpaceRecord( VJSpace*& space,
                               MicroPoint opt,
                               MicroPoint max,
                               size_t numRecords )
{
    size_t newSize = ( numRecords + 1 ) * sizeof( VJSpace );

    if ( MEMGetSizePtr( space ) <= newSize * sizeof( VJSpace ) )
        space = (VJSpace*)MEMResizePtr( (void*)&space, newSize * sizeof( VJSpace ) );
}

```

```

VJSpace*   spacePtr = space;
spacePtr += numRecords;

spacePtr->opt      = opt;
spacePtr->max      = max;
spacePtr->upperBound = (REAL)max / opt;
}

/*=====*/
/* Return the minimum line space multiplier allowed */
/* by any line in the column. */

static REAL MaxSpaceStretch( VJSpace*   space,
                             size_t     numRecords )
{
    REAL      maxStretch;

    if ( numRecords-- ) {
        maxStretch = space->upperBound;
        space++;
    }
    else
        return 0;

    for ( ; numRecords--; space++ ) {
        if ( space->upperBound < maxStretch )
            maxStretch = space->upperBound;
    }

    return maxStretch;
}

/*=====*/
/* Return the total line space expansion if every line's space is */
/* multiplied by maxStretch. */

static MicroPoint TotalSpaceStretch( VJSpace*   space,
                                     size_t     numRecords,
                                     REAL      maxStretch )
{
    REAL      totalStretch = 0;

    for ( ; numRecords--; space++ )
        totalStretch += space->opt * ( maxStretch - 1 );

    return scRoundMP( totalStretch );
}

/*=====*/
/* Return the product of the optimum line space and */
/* the stretchFactor to calculate the distance to shift a line down. */

static MicroPoint LineShift( VJSpace*   space,
                             REAL      stretchFactor,
                             short      index )
{
    MicroPoint      shift;

    /* MicroPoint * REAL */
    shift = (MicroPoint)( space[index].opt * ( stretchFactor - 1 ) );

    return shift;
}

/*=====*/
/* If flag == center, we do center vertical justification */
/* Otherwise, we do flush bottom. */

static void COLFlushBottom( scColumn*   col,
                           eDoVJ      flag )
{
    scTextline *txl;
    scTextline *lastLineH;

```



```

scTextline *tLine;
TypeSpec    spec;
MicroPoint  vDiff,
             capHeight,
             maxDepth    = LONG_MIN;
scXRect      lineRect;
RLU          capHiteRlu, d1, d2, d3;
scRLURect    rect;
Bool         vertical    = false;

COLFlushTop( col ); // remove effects of previous vj

if ( col->GetFlowdir().IsVertical() ) {
    vertical = true;
    maxDepth = LONG_MAX;
}

lastLineH = txl = col->GetFirstline();
if ( txl == NULL ) {
    return;
}

for ( ; txl != NULL; txl = LNNext( txl ) ) {
    tLine = txl;
    if ( vertical )
        maxDepth = MIN( tLine->GetOrigin().x, maxDepth );
    else {
        /* Find the depth of the last line */
        maxDepth = MAX( tLine->GetOrigin().y, maxDepth );
    }
    lastLineH = txl;
}

/* Calculate the distance between the last possible */
/* line position and our last actual position      */
if ( vertical )
    vDiff = maxDepth;
else
    vDiff = col->Depth() - maxDepth;

MicroPoint maxlead = lastLineH->MaxLead( spec );
vDiff -= CSlastLinePosition( col->GetAPPName(), spec );

if ( flag == eVJCenter ) {
    /* For center justification, cut the distance to move each
     * line in half Further adjustment must be made for the vertical
     * space occupied by the first line.
     */
    if ( ( txl = col->GetFirstline() ) != NULL ) {
        MicroPoint maxlead = txl->MaxLead( spec );

        scCachedStyle::SetFlowdir( col->GetFlowdir() );
        scCachedStyle::GetCachedStyle( spec );
        FIgetRLUFontExtents( scCachedStyle::GetCurrentCache().GetSpec(), capHiteRlu, d1, d2, d3,
rect );

        capHeight = scRoundMP( (REAL)scCachedStyle::GetCurrentCache().GetPtSize() / scBaseRLUsys
tem * capHiteRlu );

        vDiff -= CSfirstLinePosition( col->GetAPPName(), spec );
        vDiff += capHeight;

        vDiff /= 2;
    }
}

if ( vDiff != 0 ) {
    col->SetInkExtents( 0, 0, 0, 0 );

    for ( txl = col->GetFirstline(); txl != NULL; txl = LNNext( txl ) ) {
        /* Shift all the lines down by */

```

```

        /* the specified distance */
        tx1->SetVJ( vDiff );
        tx1->QueryExtents( lineRect );
        col->UnionInkExtents( lineRect );
    }
}

/*=====*/
/* Vertical justification on a column. Includes both feathering and */
/* paragraph spacing. */
/*=====*/

// TOOLBOX BEHAVIOR
// We will exceed maximum values to VJ at all costs
// In such excessive conditions, we won't use extra
// para spacing to achieve our end; we will use
// extra line spacing.
// or
// We will not exceed max values and thus may not achieve
// vertical justification
//

static const Bool        exceedMaxValues        = false;
static const Bool        extraPPspacing         = false;

static void COLFlushTopBottom( scColumn *col )
{
    VJSpace*    lineSpace = 0;
    VJSpace*    paraSpace = 0;

    scTextline* tx1;
    scTextline* tLine;
    TypeSpec    spec;
    scContUnit* lnParaH;
    scContUnit* paraH = NULL;
    short        interPara,
                 interLine,
                 lineAdj,
                 paraAdj;
    MicroPoint   vDiff,
                 currDepth = LONG_MIN,
                 currLineSpace,
                 currParaSpace,
                 maxTotalLineStretch,
                 maxTotalParaStretch,
                 adjustment,
                 lead;
    REAL          maxLineStretch,
                 maxParaStretch,
                 lineStretchFactor,
                 paraStretchFactor;
    scXRect      lineRect;
    Bool          vertical = false;

    COLFlushTop( col );           // remove effects of vj

    if ( col->GetFlowdir().IsVertical() ) {
        vertical = true;
        currDepth = LONG_MAX;
    }

    /* These handles will store arrays of structures to represent */
    /* the optimum and the maximum spacing for each line in */
    /* the column, and for each paragraph in the column. */
    /*

    try {
        lineSpace = (VJSpace*)MEMAllocPtr( sizeof( VJSpace ) * growUnits );
        paraSpace = (VJSpace*)MEMAllocPtr( sizeof( VJSpace ) * growUnits );

        interPara = interLine = 0;
        tx1 = col->GetFirstline();

```

```

currLineSpace = 0; /* These represent the current total line */
currParaSpace = 0; /* and para spacing before VJ */

for ( ; txl; txl = LNNext( txl ) ) {
    MicroPoint maxlead = txl->MaxLead( spec );
    tLine = txl;
    scCachedStyle& cs = scCachedStyle::GetCachedStyle( spec );

    if ( paraH == NULL )
        ;
    else if ( paraH == tLine->GetPara() ) {
        lead = cs.GetComputedLead( );
        /* Accumulate line space information for each line */
        InsertSpaceRecord( lineSpace, lead, cs.GetComputedMaxLead(), interLine++ );
        currLineSpace += lead;
    }
    else if ( paraH != tLine->GetPara() ) {
        lead = scCachedStyle::GetParaSpace( paraH, tLine->GetPara() );
        maxlead = scCachedStyle::GetMaxParaSpace( paraH, tLine->GetPara() );

        /* Accumulate para space information for each para */
        InsertSpaceRecord( paraSpace, lead, maxlead, interPara++ );
        currParaSpace += lead;
    }

    paraH = tLine->GetPara();

    if ( vertical )
        currDepth = MIN( tLine->GetOrigin().x, currDepth );
    else
        currDepth = MAX( tLine->GetOrigin().y, currDepth );
        /* This will tell us the */
        /* depth of the last line */
}

/* Calculate the difference between where the last line is and
 * where we want it to be
 */
if ( vertical )
    vDiff = -CSlastLinePosition( col->GetAPPName(), spec ) + currDepth;
else
    vDiff = col->Depth() - CSlastLinePosition( col->GetAPPName(), spec ) - currDepth;

/* The greatest factors by which we can */
/* multiply the space of each line and para */
maxLineStretch = MaxSpaceStretch( lineSpace, interLine );
maxParaStretch = MaxSpaceStretch( paraSpace, interPara );

/* How much space this */
/* will buy us */
maxTotalLineStretch = TotalSpaceStretch( lineSpace, interLine,
                                           maxLineStretch );
maxTotalParaStretch = TotalSpaceStretch( paraSpace, interPara,
                                           maxParaStretch );

/* How much we are currently stretching the line space */
/* and para space */
lineStretchFactor = 1;
paraStretchFactor = 1;

/* If VJ is impossible or unnecessary */
if ( currParaSpace < 0
    || currLineSpace < 0
    || ( currParaSpace == 0 && currLineSpace == 0 )
    || maxLineStretch < 0
    || maxParaStretch < 0
    || vDiff <= 0 )
{
    COLFlushTop( col );
    return;
}

if ( maxTotalParaStretch >= vDiff && currParaSpace > 0 ) {

```

```

    /* If we can do it with para spacing alone, */
    /* do it. -- REAL / MicroPoint */
    paraStretchFactor = 1 + ((REAL)vDiff) / currParaSpace;
}
else {
    if ( currParaSpace > 0 ) {
        /* Start off by stretching paragraph spacing */
        /* to the max. */
        paraStretchFactor = maxParaStretch;
        vDiff -= maxTotalParaStretch;
    }

    if ( maxTotalLineStretch >= vDiff && currLineSpace > 0 ) {
        /* If we can do remaining VJ within */
        /* max line spacing, do it -- REAL / MicroPoint */
        lineStretchFactor = 1 + ((REAL)vDiff) / currLineSpace;
    }
    else {
        if ( currLineSpace > 0 ) {
            /* Stretch line spacing to the max, */
            /* and see what's left over */
            lineStretchFactor = maxLineStretch;
            vDiff -= maxTotalLineStretch;
        }

        if ( exceedMaxValues ) {
            if ( currParaSpace > 0 && extraPPspacing ) {
                /* If extraPPspacing were true (it isn't), */
                /* we would simply increase para spacing */
                /* to cover the excess. */
                paraStretchFactor = 1 + ( ((REAL)vDiff)
                    + maxTotalParaStretch ) / currParaSpace;
            }
            /* (REAL + MicroPoint) / MicroPoint */
            else {
                /* Spread remaining space evenly over all remaining
                 * lines, including both inter line and inter para
                 * spacing.
                 */
                /* Some care is requires to do it evenly. */
                MicroPoint totalParaSpace
                    = currParaSpace ? currParaSpace+maxTotalParaStretch:0;

                MicroPoint totalLineSpace
                    = currLineSpace ? currLineSpace+maxTotalLineStretch:0;

                MicroPoint totalSpace
                    = totalParaSpace + totalLineSpace;

                MicroPoint paraDiff
                    = scRoundMP( ((REAL)totalParaSpace) / totalSpace * vDiff );

                MicroPoint lineDiff
                    = scRoundMP( ((REAL)totalLineSpace) / totalSpace * vDiff );

                /* REAL / MicroPoint / MicroPoint */

                if ( currParaSpace )
                    paraStretchFactor = 1 + ( ((REAL)paraDiff)
                        + maxTotalParaStretch ) / currParaSpace;
                if ( currLineSpace )
                    lineStretchFactor = 1 + ( ((REAL)lineDiff)
                        + maxTotalLineStretch ) / currLineSpace;
                /* ( REAL + MicroPoint ) / MicroPoint */
            }
        }
    }
}

adjustment = 0;
lineAdj = paraAdj = 0;
paraH = NULL;

```

```

col->SetInkExtents( 0, 0, 0, 0 );

for ( txl = col->GetFirstline(); txl; txl = LNNext( txl ) ) {
    lnParaH = txl->GetPara( );

    /* Shift each line down the appropriate amount */

    if ( paraH == NULL )
        txl->SetVJ( 0 );
    else if ( paraH == txl->GetPara() ) {
        /* Add line space */
        adjustment += LineShift( lineSpace, lineStretchFactor, lineAdj );
        txl->SetVJ( adjustment );
        lineAdj++;
    }
    else if ( paraH != txl->GetPara() ) {
        /* Add para space */
        adjustment += LineShift( paraSpace, paraStretchFactor, paraAdj );
        txl->SetVJ( adjustment );
        paraAdj++;
    }
    paraH = lnParaH;
    txl->QueryExtents( lineRect );
    col->UnionInkExtents( lineRect );
}
}
catch( ... ) {
    MEMFreePtr( lineSpace );
    MEMFreePtr( paraSpace );
    throw;
}
MEMFreePtr( lineSpace );
MEMFreePtr( paraSpace );
}

/*=====*/
/* Shove the lines to the top */
static void COLFlushTop( scColumn* col )
{
    for ( scTextline* txl = col->GetFirstline(); txl; txl = LNNext( txl ) )
        txl->RemoveVJ( );
}

/*=====*/

void scColumn::SetDepthNVJ( MicroPoint    dimension,
                           scRedisplList* redisplList )
{
    scXRect lineDamage;

    if ( Marked( scINVALID ) )
        LimitDamage( redisplList, scReformatTimeSlice );

    if ( fFlowDir.IsHorizontal() )
        SetDepth( dimension );
    else
        SetWidth( dimension );

    scRedisplayStoredLine rdl( GetLinecount() );
    rdl.SaveLineList( this );
    VertJustify( );
    rdl.LineListChanges( this, lineDamage, redisplList );
}

/*=====*/
/* align the text lines in the column,
 * this function just serves as a dispatcher
 */

```

```

void scColumn::VertJustify( )
{
    eVertJust      attributes      = GetVertJust();
    eColShapeType  colShape        = GetShapeType();

    if ( ! ( colShape == eNoShape || ( colShape & eFlexShape ) ) )
        COLFlushTop( this );
    else {
        switch ( attributes ) {
            case eVertJustified:
                if ( !GetNext() ) {
                    /* If this is the stream's last column, don't VJ */
                    /* unless force VJ is set. If it isn't the last */
                    /* column, fall through to the next case to VJ. */
                    COLFlushTop( this );    // remove effects of vj
                    break;
                }
                // let this fall thru

            case eVertForceJustify:
                COLFlushTopBottom( this );
                break;

            case eVertBottom:
                COLFlushBottom( this, eVJBottom );
                break;

            case eVertCentered:
                COLFlushBottom( this, eVJCenter );
                break;

            default:
                case eVertTop:
                    COLFlushTop( this );
                    break;
        }
    }
}

/*=====*/
/* Determine the number of lines in a column */
ushort scColumn::GetLinecount( ) const
{
    scTextline* txl;
    ushort      lineCount;

    for ( lineCount = 0, txl = GetFirstline(); txl; txl = txl->GetNext() )
        lineCount++;
    return lineCount;
}

/*=====*/
/* The functions that follow are used to keep track of which lines */
/* move during VJ, to minimize repainting. If any of it fails due */
/* lack of memory, VJ is not jeopardized, but everything will end */
/* up being repainted. */
/*=====*/

scRedisplayStoredLine::scRedisplayStoredLine( int lines ) :
    fStoredData( 0 ),
    fStoredLines( 0 ),
    fUsingStoredData( false ),
    fData( 0 ),
    fNumItems( 0 )
{
    LineListInit( lines );
}

/*=====*/

```

```

scRedisplayStoredLine::~scRedisplayStoredLine( )
{
    delete [] fStoredData, fStoredData = NULL;
    fStoredLines = 0;
}

/*-----*/

void scRedisplayStoredLine::LineListInit( int lines )
{
    fUsingStoredData = false;

    fData = 0;
    fNumItems = 0;

#ifdef MWERKS_NEW_ARRAY_PROBLEM
    fStoredData = SCNEW scTextline[ lines ];
#else
    fStoredData = new scTextline[ lines ];
#endif

    fStoredLines = (short)lines;
}

/*-----*/

void scRedisplayStoredLine::LineListFini( )
{
    ushort i;
    for ( i = 0; i < fStoredLines; i++ )
        fStoredData[i].InitForReuse( 0 );
    delete [] fStoredData, fStoredData = NULL;
    fStoredLines = 0;
}

/*-----*/
/* Save an image of the lines in a column to determine repainting
   at the completion of reformatting
*/
void scRedisplayStoredLine::SaveLineList( scColumn* col )
{
    scTextline* txlCopy;
    scTextline* txl;
    ushort lines;

    lines = col->GetLinecount( );

    fNumItems = lines;
    fOrgExtents = col->GetInkExtents();

    if ( lines ) {
        // determine if we are using the stored lines ( about 200 )
        // or do we dynamically allocate a list - if more than 200 lines
        // - which should be almost never
        //
        if ( fStoredData && lines < fStoredLines ) {
            fUsingStoredData = true;
            fData = fStoredData;
        }
        else {
            fData = new scTextline[ lines ];
            fUsingStoredData = false;
        }

        // copy current state to the list of lines
        txl = col->GetFirstline();
        for ( txlCopy = fData ; lines--; txl = LNNext( txl ), txlCopy++ ) {
            *txlCopy = *txl;
        }
    }

#ifdef 0
    scXRect xrect;
    txl->QueryExtents( xrect, 1 );
#endif
}

```

```

        txlCopy->SetInkExtents( xrect );
#endif
    }
    scAssert( txl == NULL );
}
else
    fData = NULL;
}

/*=====*/
// compare the list of saved lines with the current column lines and
// determine the repainting that needs to be done

void scRedisplayStoredLine::LineListChanges( scColumn*      col,
                                              const scXRect& oldLineDamage,
                                              scRedisplist*  redisplist )
{
    scTextline* txlOrg;
    scTextline* txl;
    scXRect      lineDamage( oldLineDamage );
    ushort      lines      = col->GetLinecount();

    scStreamChangeInfo streamChange;

    streamChange = gStreamChangeInfo;

    if ( fData == NULL ) {
        // redraw the entire column
        if ( redisplist ) {
            col->QueryMargins( fOrgExtents );
            redisplist->AddColumn( col, fOrgExtents );
        }
        col->Unmark( scREPAINT );
    }
    else {
        txl = col->GetFirstline();

        //
        // compare old lines and new lines and where they differ
        // mark that area to be repainted
        //
        for ( txlOrg = fData; lines && fNumItems; txl = LNNNext( txl ), txlOrg++ ) {
            lines--;
            fNumItems--;
            if ( !txl->Compare( txlOrg, streamChange ) ) {
                // handle old line position now
                scXRect xrect,
                    xrect2;
                txl->QueryExtents( xrect, 1 );
                txlOrg->QueryExtents( xrect2, 1 );
                lineDamage.Union( xrect );
                lineDamage.Union( xrect2 );
                txl->Unmark( scREPAINT );
            }
        }

        // fData ran out first, mark the rest of the new lines
        for ( ; lines--; txl = LNNNext( txl ) ) {
            scXRect xrect;
            txl->QueryExtents( xrect, 1 );
            lineDamage.Union( xrect );
        }

        // current lines ran out first
        for( ; fNumItems--; txlOrg++ ) {
            scXRect xrect;
            txlOrg->QueryExtents( xrect, 1 );
            lineDamage.Union( xrect );
        }

        if ( redisplist )

```

[illegible]

File: SCCOLUM3.C

\$Header: /Projects/Toolbox/ct/Sccolumn3.cpp 3 5/30/97 8:45a Wmanis \$

Contains: Contains the code to allocate lines for containers and
other miscellaneous code.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

*****/

```
#include "scpubobj.h"
#include "sccolumn.h"
#include "scstcach.h"
#include "scglobda.h"
#include "sccallbk.h"
#include "scmem.h"
#include "scparagr.h"
#include "scregion.h"
#include "sctextli.h"
#include "screfdat.h"
```

```
/* ===== */
// Translate the lines
```

```
void scColumn::TranslateLines( const scMuPoint& trans )
{
    scTextline* txl;

    for ( txl = fFirstline; txl; txl = txl->GetNext() ) {
        scAssert( txl->fOrigin.x + trans.x >= 0 );
        txl->Translate( trans );
    }
}
```

```
/* ===== */
/* reposition or realign the lines in this column which is probably a
 * flex column
 */
```

```
void scColumn::RepositionLines( )
{
    scTextline *txl;
    MicroPoint measure;

    if ( GetFlowdir().IsHorizontal() )
        measure = Width( );
    else
        measure = Depth();

    for ( txl = fFirstline; txl; txl = txl->GetNext() )
        txl->Reposition( measure );
}
```

```
/* ===== */
// set the column as the active container in the reformatting cache
```

```
void scCOLRefData::SetActive( scColumn* col )
```

```

{
    fCol = col;
    if ( col )
        fPData.SetFlowDir( col->GetFlowdir() );
    else
        fPData.SetFlowDir( scFlowDir( ) );
}

/* ===== */
// free lines marks as invalid and collect their damaged area

void scCOLRefData::FreeInvalidLines( void )
{
    scTextline* txl;
    scTextline* nextTxl;

    for ( txl = fCol->GetFirstline(); txl; txl = nextTxl ) {
        nextTxl = txl->GetNext();
        if ( txl->Marked( scINVALID ) )
            txl->Delete( fLineDamage );
        else if ( txl->Marked( scREPAINT ) ) {
            scXRect damage;
            txl->QueryExtents( damage, 1 );
            fLineDamage.Union( damage );
            txl->Unmark( scREPAINT );
        }
    }
}

/* ===== */
// save the linelist for damage determination in formatting

void scCOLRefData::SaveLineList( )
{
    scTextline* txl;

    for ( txl = fCol->GetFirstline(); txl; txl = txl->GetNext() ) {
        if ( txl->Marked( scREPAINT ) ) {
            scXRect xrect;
            txl->QueryExtents( xrect, 1 );
            fLineDamage.Union( xrect );
            txl->Marked( scREPAINT );
        }
    }

    fSavedLineState.SaveLineList( fCol );
}

/* ===== */
// initialize the the data structures to perform linebreaking in a column,
// this is primarily used for irregular run-arounds

Bool scCOLRefData::COLInit( scColumn* col, scContUnit* p )
{
    fCol = col;
    fLineDamage.Invalidate();

    FreeInvalidLines( );

    SCDebugTrace( 2, scString( "\tCOLStartReformat: col 0x%08x %d\n" ), fCol, fCol->GetCount( ) );
    scVertex*      startV;
    scContUnit*    prevPara;
    PrevParaData   prevParaData;

    prevParaData.lastLineH = NULL;
    prevParaData.lastSpec.clear();

    // SCDebugTrace( 2, scString( "COLStartReformat %d" ), col->fColumnCount );

    if ( !fCol->DamOpen() )
        return false;
}

```

```

#if SCDEBUG > 1
    if ( fCol->GetPrev() )
        scAssert( !fCol->GetPrev()->Marked( scLAYACTIVE ) );
    scAssert( fCol && ! fCol->Marked( scLAYACTIVE ) );
#endif

    fCol->Mark( scLAYACTIVE );

    ggcS.theActiveColH = fCol;

    SetActive( fCol );

    scCachedStyle::SetFlowdir( fCol->GetFlowdir() );

    // set these to defaults
    fSavedPrevEnd.Set( LONG_MIN, FIRST_LINE_POSITION );

    // now check to see if we are starting reformatting in the
    // middle of the column, if we are we should set the prevbaseline up
    prevPara = p->GetPrev();
    if ( prevPara ) {
        scTextline* txl = prevPara->GetLastline( );
        if ( txl && txl->GetColumn() == fCol ) {
            scColumn* tCol = fCol;
            scLEADRefData lead;
            MicroPoint baseline = fSavedPrevEnd.y;
            p->LocateFirstLine( *this, p->SpecAtStart(), tCol, baseline, lead, prevParaData );
            scAssert( tCol == fCol );
        }

        /* If this fails, no problem. COLLineListChanges */
        /* will simply repaint ALL lines. */
        SaveLineList( );

        SetRegion( 0 );

        switch ( fCol->GetShapeType() ) {
            default:
                break;
            case eFlexShape:
            case eHorzFlex:
                if ( fCol->GetFlowdir().IsVertical() ) {
                    scTextline* txl = fCol->GetFirstline();
                    if ( txl ) {
                        MicroPoint position = txl->GetOrigin().x;
                        TypeSpec ts = txl->SpecAtStart( );
                        MicroPoint firstlinepos = CSfirstLinePosition( fCol->GetAPPName(), ts );

                        scMuPoint trans( mpInfinity - position - firstlinepos, 0 );
                        fCol->TranslateLines( trans );
                        fCol->SetWidth( mpInfinity );
                    }
                }
                break;
            case eVertShape:
                scCachedStyle::GetCurrentCache().SetRunAroundBorder( CSrunaroundBorder( fCol->GetAPPName() ), scCachedStyle::GetCurrentCache().GetSpec() );
                startV = (scVertex *)MEMLockHnd( fCol->GetVertList() );

                try {
                    fRgnH = NewHRgn( scSliverSize() );
                    PolyHRgn( fRgnH, startV );
                }
                catch ( ... ) {
                    DisposeHRgn( fRgnH ), fRgnH = 0;
                    MEMUnlockHnd( fCol->GetVertList() );
                    throw;
                }

                MEMUnlockHnd( fCol->GetVertList() );

```

```

    InsetHRgn( fRgnH, scCachedStyle::GetCurrentCache().GetRunAroundBorder(), scCachedStyle::
GetCurrentCache().GetRunAroundBorder(), true );

    fRgn = (HRgn*)MEMLockHnd( fRgnH );

    break;

case eRgnShape:
    scCachedStyle::GetCurrentCache().SetRunAroundBorder( CRunaroundBorder( fCol->GetAppName
()), scCachedStyle::GetCurrentCache().GetSpec() );

    try {
        fRgnH = NewHRgn( RGNSliverSize( fCol->GetRgn() ) );

        CopyHRgn( fRgnH, fCol->GetRgn() );
        InsetHRgn( fRgnH, scCachedStyle::GetCurrentCache().GetRunAroundBorder(), scCachedSty
le::GetCurrentCache().GetRunAroundBorder(), true );
    }
    catch ( ... ) {
        DisposeHRgn( fRgnH ), fRgnH = 0;
        throw;
    }

    fRgn = (HRgn *)MEMLockHnd( fRgnH );
    break;
}

fPData.fInitialLine.fBaseline = FIRST_LINE_POSITION;
fPData.fComposedLine.fBaseline = FIRST_LINE_POSITION;

return true;
}

/* ----- */
// close out the data structures when finished line breaking in a column

void ScCOLRefData::COLFini( Bool finished )
{
    if ( fCol ) {
        scXRect extents;

        scAssert( fCol->Marked( scLAYACTIVE ) && fCol == GetActive() );
        fCol->Unmark( scLAYACTIVE );

        // SCDebugTrace( 2, scString( "COLEndReformat %d" ), col->fColumnCount );

        ggcS.theActiveColH = NULL;

        switch ( fCol->GetShapeType() ) {
            default:
                fCol->VertJustify( );
                break;

            case eVertShape:
            case eRgnShape:
                fCol->VertJustify( );
                if ( fRgnH ) {
                    MEMUnlockHnd( fRgnH );
                    DisposeHRgn( fRgnH ), fRgnH = NULL;
                }
                break;

            case eHorzFlex:
            case eFlexShape:
            case eVertFlex:
                fCol->QueryMargins( extents );
                if ( fCol->GetFlowdir().IsHorizontal() ) {
                    if ( fCol->GetShapeType() & eVertFlex )
                        fCol->SetDepth( extents.y2 );
                    if ( fCol->GetShapeType() & eHorzFlex ) {
                        fCol->SetWidth( extents.x2 );
                        fCol->RepositionLines();
                    }
                }
                break;
        }
    }
}

```

```

    }
    }
    else {
        if ( fCol->GetShapeType() & eHorzFlex ) {
            scMuPoint trans( extents.Width() - fCol->Width(), 0 );
            fCol->TranslateLines( trans );

            fCol->SetWidth( extents.Width() );
        }
        if ( fCol->GetShapeType() & eVertFlex ) {
            fCol->SetDepth( extents.y2 );
            fCol->RepositionLines();
        }
    }
    fCol->VertJustify();
    break;
}

fSavedLineState.LineListChanges( fCol, fLineDamage, fRedisplist );

if ( finished )
    fCol->Unmark( scINVALID | scREALIGN );

SCDebugTrace( 2, scString( "\tCOLEndReformat: col 0x%08x %d\n" ), fCol, fCol->GetCount( ) );
}

/* ===== */
// add all lines that need to be repainted to the repaint rect
scXRect& scColumn::RepaintExtent( scXRect& repaintExtents )
{
    scXRect    lineExtents;
    scTextline* txl;

    repaintExtents.Invalidate();
    for ( txl = GetFirstline(); txl; txl = txl->GetNext() ) {
        if ( txl->Marked( scREPAINT ) ) {
            txl->QueryExtents( lineExtents, 1 );
            if ( lineExtents.Width() == 0 )
                lineExtents.x2 = lineExtents.x1 + 1;
            repaintExtents.Union( lineExtents );
            txl->Marked( scREPAINT );
        }
    }
    Marked( scREPAINT );
    return repaintExtents;
}

/* ===== */
// set the column's vertical justification attribute

void scColumn::SetVJ( eVertJust attr )
{
    if ( GetVertJust() != attr )
        Mark( scREALIGN );
    SetVertJust( attr );
}

/* ===== */
// this does the actual space allocation within the column

Bool scColumn::GetStrip2( scLINERefData&    lineData,
                          int               breakType,
                          scCOLRefData&     colRefData )
{
    {
        scXRect    tryRect;
        MicroPoint tryX,
                   tryY,
                   colWidth = Width( ),
                   colDepth = Depth(),
                   firstLinePosition;
    }
}

```

```

Bool      firstLine = false;
int       shapeType;

    // the specs have been properly initied so the block
    // indent values should be correct
scAssert( this == colRefData.GetActive() );

    // we are in an overflow condidtion
if ( lineData.fOrg.y == LONG_MIN )
    return false;
if ( breakType == eColumnBreak )
    return false;

lineData.fColShapeType = GetShapeType();
shapeType              = GetShapeType();

    // We ran into a memory error in COLStartReformat; just use rectangle shape.
if ( fRgnH && colRefData.fRgnH == NULL )
    lineData.fColShapeType = eNoShape;

if ( lineData.IsVertical() ) {
    colWidth      = Depth();
    colDepth      = Width();
    switch ( GetShapeType() ) {
        case eVertFlex:
            shapeType = eHorzFlex;
            break;
        case eHorzFlex:
            shapeType = eVertFlex;
            break;
    }
}

colRefData.fPrevEnd      = colRefData.fSavedPrevEnd;

switch ( shapeType ) {
    default:
        case eNoShape:
            lineData.fOrg.x      = scCachedStyle::GetParaStyle().GetLeftBlockIndent();
            lineData.fMeasure    = colWidth - scCachedStyle::GetParaStyle().GetLeftBlockIndent();
            if ( lineData.fOrg.y == FIRST_LINE_POSITION )
                lineData.fOrg.y = CSfirstLinePosition( GetAPPName(), scCachedStyle::GetCurrentCache(
                    ).GetSpec() );
            else
                lineData.fOrg.y += lineData.fInitialLead.GetLead();

            return lineData.fOrg.y <= colDepth - CSlastLinePosition( GetAPPName(), scCachedStyle::Ge
                tCurrentCache().GetSpec() );

        case eVertShape:
        case eRgnShape:
            tryRect.Set( 0, 0, MAX( scCachedStyle::GetParaStyle().GetMinMeasure(), colRefData.fRgn->
                fVertInterval ), lineData.fLogicalExtents.Depth() );

            if ( lineData.fOrg.y == FIRST_LINE_POSITION ) {
                firstLine      = true;
                firstLinePosition = CSfirstLinePosition( GetAPPName(), scCachedStyle::GetCurrentCa
                    che().GetSpec() );
                lineData.fOrg.y      = colRefData.fRgn->FirstLinePos( firstLinePosition, lineData.fIn
                    itialLead.GetLead() );
                tryX              = colRefData.fPrevEnd.x;
                tryY              = lineData.fOrg.y - firstLinePosition;
                tryRect.y2        = firstLinePosition + CSlastLinePosition( GetAPPName(), scCached
                    Style::GetCurrentCache().GetSpec() );
                colRefData.SetFirstlinePos( lineData.fOrg.y );
                colRefData.SetFirstSpec( scCachedStyle::GetCurrentCache().GetSpec() );
            }
            else {
                if ( lineData.fOrg.y == colRefData.GetFirstlinePos() ) {
                    firstLine      = true;
                    firstLinePosition = CSfirstLinePosition( GetAPPName(), colRefData.GetFirstSpec
                        ( ) );
                    lineData.fOrg.y      = colRefData.fRgn->FirstLinePos( firstLinePosition, lineData

```

```

.fInitialLead.GetLead() );
    tryX          = colRefData.fPrevEnd.x;
    tryY          = lineData.fOrg.y - firstLinePosition;
    tryRect.y2    = firstLinePosition + CSlastLinePosition( GetAPPName(), scCa
chedStyle::GetCurrentCache().GetSpec() );
    }
    else
        tryY = lineData.fOrg.y + lineData.fLogicalExtents.y1;

    if ( lineData.IsHorizontal() ) {
        if ( colRefData.fPrevEnd.y == lineData.fOrg.y )
            tryX = colRefData.fPrevEnd.x;
        else
            tryX = LONG_MIN;
    }
    else {
        if ( ( colDepth - colRefData.fPrevEnd.x ) == lineData.fOrg.y )
            tryX = colRefData.fPrevEnd.y;
        else
            tryX = LONG_MIN;
    }
}

colRefData.fRgn->SectRect( tryRect, tryY, tryX, lineData.fInitialLead.GetLead() );

ePos() ) {
    // this is here to fix the smi bug 1538 - given that we are using approximations

    // alot in regions this may be an insufficient fix for other issues that smi
    // may raise, but since we are using approximations i have no way of reliably
    // predicting these issues
    scXRect rgnXRect( colRefData.fRgn->fOrigBounds );
    scXRect tryXRect( tryRect );

    if ( !rgnXRect.Contains( tryXRect ) )
        tryRect.x2 = tryRect.x1;
}

if ( tryRect.Width() == 0 )
    return false;
else {
    if ( firstLine == true )
        lineData.fOrg.y = tryRect.y1 + firstLinePosition;
    else
        lineData.fOrg.y = tryRect.y1 - lineData.fLogicalExtents.y1;

    if ( lineData.fOrg.y > RGNMaxDepth( colRefData.fRgnH ) )
        return false;

#ifdef LEFTBLOCKINDENT
    if ( tryRect.x < 0 )
        lineData.fOrg.x = tryRect.x + gfmS.GetLeftBlockIndent();
    else
        lineData.fOrg.x = MAX( tryRect.x, gfmS.GetLeftBlockIndent() );
#else
    lineData.fOrg.x          = tryRect.x1 + scCachedStyle::GetParaStyle().GetLeftBlockIn
dent();
#endif

    if ( lineData.fOrg.x != tryRect.x1 )
        lineData.fMeasure = tryRect.Width() + ( tryRect.x1 - lineData.fOrg.x );
    else
        lineData.fMeasure = tryRect.Width();

    return true;
}
break; /*NOTREACHED*/

case eVertFlex:
    lineData.fOrg.x          = scCachedStyle::GetParaStyle().GetLeftBlockIndent();
    if ( lineData.fOrg.y == FIRST_LINE_POSITION )
        lineData.fOrg.y = CSfirstLinePosition( GetAPPName(), scCachedStyle::GetCurrentCache(

```



```

).GetSpec() );
    else
        lineData.fOrg.y += lineData.fInitialLead.GetLead();

    lineData.fMeasure = colWidth - scCachedStyle::GetParaStyle().GetLeftBlockIndent();
    return lineData.fMeasure > 0;

    case eHorzFlex:
        lineData.fOrg.x = scCachedStyle::GetParaStyle().GetLeftBlockIndent();
        if ( lineData.fOrg.y == FIRST_LINE_POSITION )
            lineData.fOrg.y = CSfirstLinePosition( GetAPPName(), scCachedStyle::GetCurrentCache(
).GetSpec() );
        else
            lineData.fOrg.y += lineData.fInitialLead.GetLead();
        lineData.fMeasure = HorzFlexMeasure;

        return lineData.fOrg.y <= colDepth - CSlastLinePosition( GetAPPName(), scCachedStyle::Ge
tCurrentCache().GetSpec() );

    case eFlexShape:
        lineData.fOrg.x = scCachedStyle::GetParaStyle().GetLeftBlockIndent();
        if ( lineData.fOrg.y == FIRST_LINE_POSITION )
            lineData.fOrg.y = CSfirstLinePosition( GetAPPName(), scCachedStyle::GetCurrentCache(
).GetSpec() );
        else
            lineData.fOrg.y += lineData.fInitialLead.GetLead();
        lineData.fMeasure = HorzFlexMeasure;

        return true;
    }
    /*NOTREACHED*/
    return false;
}

// ===== */
// allocate geometry using args
Bool scColumn::GetStrip( scLINERefData& lineData,
                        int breakType,
                        scCOLRefData& colRefData )
{
    Bool doit;

    // since the get strip logic uses regions and can only really
    // deal in one dimension we need to convert the coordinate
    // system of the used variables as we go in and out of the
    // get strip code - refer to the discussion of coordinate
    // systems in the Toolbox Concept doc
    if ( lineData.IsVertical() )
        lineData.fLogicalExtents.FourthToThird( 0 );

    lineData.fOrg.y = lineData.fBaseline;

    doit = GetStrip2( lineData, breakType, colRefData );

    lineData.fBaseline = lineData.fOrg.y;

    if ( lineData.IsVertical() ) {
        lineData.fOrg.ThirdToFourth( Width() );
        lineData.fLogicalExtents.ThirdToFourth( 0 );
    }

    return doit;
}

/* ===== */
// allocate geomtry using cached values
Bool scCOLRefData::AllocGeometry( void )
{
    Bool doit;

```

[illegible]

$\Gamma_{\text{eff}}^{(1)}(\mathbf{p})$ is the effective one-loop self-energy, $\Gamma_{\text{eff}}^{(2)}(\mathbf{p})$ is the effective two-loop self-energy, and $\Gamma_{\text{eff}}^{(3)}(\mathbf{p})$ is the effective three-loop self-energy. The effective self-energies are calculated using the ϵ -expansion, and the results are compared with the results of the ϵ -expansion.

File: SCCTYPE.C

\$Header: /Projects/Toolbox/ct/SCCTYPE.CPP 2 5/30/97 8:45a Wmanis \$

Contains: Character types.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/
#include "sccharex.h"
#include "scctype.h"

unsigned short sc_CharType[258] = {
    0, /* to let us use -1 as an index */
    0, /* 0x00 0 " " */
    0, /* 0x01 1 " " */
    0, /* 0x02 2 " " */
    0, /* 0x03 3 " " */
    0, /* 0x04 4 " " */
    0, /* 0x05 5 " " */
    0, /* 0x06 6 " " */
    sc_ASCII|sc_SPACE, /* 0x07 7 "indent space" */
    0, /* 0x08 8 " " */
    sc_ASCII|sc_SPACE, /* 0x09 9 "\t" (HT) tab key */
    sc_ASCII|sc_SPACE, /* 0x0a 10 "\n" (LF) line feed */
    sc_ASCII|sc_SPACE, /* 0x0b 11 "\l" (VT) vertical tab */
    0, /* 0x0c 12 " " */
    sc_ASCII|sc_SPACE, /* 0x0d 13 "\r" (CR) return key */
    0, /* 0x0e 14 " " */
    0, /* 0x0f 15 " " */
    0, /* 0x10 16 " " */
    0, /* 0x11 17 " " */
    sc_ASCII, /* 0x12 18 "paraEnd" */
    sc_ASCII|sc_SPACE, /* 0x13 19 "quad center" */
    sc_ASCII|sc_SPACE, /* 0x14 20 "quad left" */
    sc_ASCII|sc_SPACE, /* 0x15 21 "quad right" */
    sc_ASCII|sc_SPACE, /* 0x16 22 "quad just" */
    sc_ASCII|sc_SPACE, /* 0x17 23 " " fix abs space */
    sc_ASCII|sc_SPACE, /* 0x18 24 " " fix rel space */
    sc_ASCII|sc_SPACE, /* 0x19 25 " " fill space */
    sc_ASCII, /* 0x1a 26 " " no break hyphen */
    sc_ASCII|sc_SPACE, /* 0x1b 27 " " discretionary hyphen */
    sc_ASCII|sc_SPACE, /* 0x1c 28 " " figure space */
    sc_ASCII|sc_SPACE, /* 0x1d 29 " " thin space */
    sc_ASCII|sc_SPACE, /* 0x1e 30 " " en space */
    sc_ASCII|sc_SPACE, /* 0x1f 31 " " em space */
    // [0020] SPACE
    // [0021] EXCLAMATION_MARK
    // [0022] QUOTATION_MARK
    // [0023] NUMBER_SIGN
    // [0024] DOLLAR_SIGN
    // [0025] PERCENT_SIGN
    // [0026] AMPERSAND
    // [0027] APOSTROPHE
    // [0028] LEFT_PARENTHESIS
    // [0029] RIGHT_PARENTHESIS
    // [002A] ASTERISK
    // [002B] PLUS_SIGN
    // [002C] COMMA

```

```

sc_ASCII|sc_PUNC, // [002D] HYPHEN-MINUS
sc_ASCII|sc_PUNC, // [002E] FULL_STOP
sc_ASCII|sc_SYMBOL, // [002F] SOLIDUS
sc_ASCII|sc_DIGIT, // [0030] DIGIT_ZERO
sc_ASCII|sc_DIGIT, // [0031] DIGIT_ONE
sc_ASCII|sc_DIGIT, // [0032] DIGIT_TWO
sc_ASCII|sc_DIGIT, // [0033] DIGIT_THREE
sc_ASCII|sc_DIGIT, // [0034] DIGIT_FOUR
sc_ASCII|sc_DIGIT, // [0035] DIGIT_FIVE
sc_ASCII|sc_DIGIT, // [0036] DIGIT_SIX
sc_ASCII|sc_DIGIT, // [0037] DIGIT_SEVEN
sc_ASCII|sc_DIGIT, // [0038] DIGIT_EIGHT
sc_ASCII|sc_DIGIT, // [0039] DIGIT_NINE
sc_ASCII|sc_PUNC, // [003A] COLON
sc_ASCII|sc_PUNC, // [003B] SEMICOLON
sc_ASCII|sc_SYMBOL, // [003C] LESS-THAN_SIGN
sc_ASCII|sc_SYMBOL, // [003D] EQUALS_SIGN
sc_ASCII|sc_SYMBOL, // [003E] GREATER-THAN_SIGN
sc_ASCII|sc_PUNC, // [003F] QUESTION_MARK
sc_ASCII|sc_SYMBOL, // [0040] COMMERCIAL_AT
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0041] LATIN_CAPITAL_LETTER_A
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0042] LATIN_CAPITAL_LETTER_B
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0043] LATIN_CAPITAL_LETTER_C
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0044] LATIN_CAPITAL_LETTER_D
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0045] LATIN_CAPITAL_LETTER_E
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0046] LATIN_CAPITAL_LETTER_F
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0047] LATIN_CAPITAL_LETTER_G
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0048] LATIN_CAPITAL_LETTER_H
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0049] LATIN_CAPITAL_LETTER_I
sc_ASCII|sc_ALPHA|sc_UPCASE, // [004A] LATIN_CAPITAL_LETTER_J
sc_ASCII|sc_ALPHA|sc_UPCASE, // [004B] LATIN_CAPITAL_LETTER_K
sc_ASCII|sc_ALPHA|sc_UPCASE, // [004C] LATIN_CAPITAL_LETTER_L
sc_ASCII|sc_ALPHA|sc_UPCASE, // [004D] LATIN_CAPITAL_LETTER_M
sc_ASCII|sc_ALPHA|sc_UPCASE, // [004E] LATIN_CAPITAL_LETTER_N
sc_ASCII|sc_ALPHA|sc_UPCASE, // [004F] LATIN_CAPITAL_LETTER_O
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0050] LATIN_CAPITAL_LETTER_P
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0051] LATIN_CAPITAL_LETTER_Q
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0052] LATIN_CAPITAL_LETTER_R
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0053] LATIN_CAPITAL_LETTER_S
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0054] LATIN_CAPITAL_LETTER_T
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0055] LATIN_CAPITAL_LETTER_U
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0056] LATIN_CAPITAL_LETTER_V
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0057] LATIN_CAPITAL_LETTER_W
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0058] LATIN_CAPITAL_LETTER_X
sc_ASCII|sc_ALPHA|sc_UPCASE, // [0059] LATIN_CAPITAL_LETTER_Y
sc_ASCII|sc_ALPHA|sc_UPCASE, // [005A] LATIN_CAPITAL_LETTER_Z
sc_ASCII|sc_SYMBOL, // [005B] LEFT_SQUARE_BRACKET
sc_ASCII|sc_SYMBOL, // [005C] REVERSE_SOLIDUS
sc_ASCII|sc_SYMBOL, // [005D] RIGHT_SQUARE_BRACKET
sc_ASCII|sc_SYMBOL, // [005E] CIRCUMFLEX_ACCENT
sc_ASCII|sc_SYMBOL, // [005F] LOW_LINE
sc_ASCII|sc_ACCENT, // [0060] GRAVE_ACCENT
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0061] LATIN_SMALL_LETTER_A
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0062] LATIN_SMALL_LETTER_B
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0063] LATIN_SMALL_LETTER_C
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0064] LATIN_SMALL_LETTER_D
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0065] LATIN_SMALL_LETTER_E
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0066] LATIN_SMALL_LETTER_F
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0067] LATIN_SMALL_LETTER_G
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0068] LATIN_SMALL_LETTER_H
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0069] LATIN_SMALL_LETTER_I
sc_ASCII|sc_ALPHA|sc_LOCASE, // [006A] LATIN_SMALL_LETTER_J
sc_ASCII|sc_ALPHA|sc_LOCASE, // [006B] LATIN_SMALL_LETTER_K
sc_ASCII|sc_ALPHA|sc_LOCASE, // [006C] LATIN_SMALL_LETTER_L
sc_ASCII|sc_ALPHA|sc_LOCASE, // [006D] LATIN_SMALL_LETTER_M
sc_ASCII|sc_ALPHA|sc_LOCASE, // [006E] LATIN_SMALL_LETTER_N
sc_ASCII|sc_ALPHA|sc_LOCASE, // [006F] LATIN_SMALL_LETTER_O
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0070] LATIN_SMALL_LETTER_P
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0071] LATIN_SMALL_LETTER_Q
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0072] LATIN_SMALL_LETTER_R
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0073] LATIN_SMALL_LETTER_S
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0074] LATIN_SMALL_LETTER_T
sc_ASCII|sc_ALPHA|sc_LOCASE, // [0075] LATIN_SMALL_LETTER_U

```

```

sc_ASCII|sc_ALPHA|sc_LOCALE, // [0076] LATIN_SMALL_LETTER_V
sc_ASCII|sc_ALPHA|sc_LOCALE, // [0077] LATIN_SMALL_LETTER_W
sc_ASCII|sc_ALPHA|sc_LOCALE, // [0078] LATIN_SMALL_LETTER_X
sc_ASCII|sc_ALPHA|sc_LOCALE, // [0079] LATIN_SMALL_LETTER_Y
sc_ASCII|sc_ALPHA|sc_LOCALE, // [007A] LATIN_SMALL_LETTER_Z
sc_ASCII|sc_SYMBOL, // [007B] LEFT_CURLY_BRACKET
sc_ASCII|sc_SYMBOL, // [007C] VERTICAL_LINE
sc_ASCII|sc_SYMBOL, // [007D] RIGHT_CURLY_BRACKET
sc_ASCII|sc_SYMBOL, // [007E] TILDE

0, // 0x7f 127
0, // 0x80 128
0, // 0x81 129
0, // 0x82 130
0, // 0x83 131
0, // 0x84 132
0, // 0x85 133
0, // 0x86 134
0, // 0x87 135
0, // 0x88 136
0, // 0x89 137
0, // 0x8a 138
0, // 0x8b 139
0, // 0x8c 140
0, // 0x8d 141
0, // 0x8e 142
0, // 0x8f 143
0, // 0x90 144
0, // 0x91 145
0, // 0x92 146
0, // 0x93 147
0, // 0x94 148
0, // 0x95 149
0, // 0x96 150
0, // 0x97 151
0, // 0x98 152
0, // 0x99 153
0, // 0x9a 154
0, // 0x9b 155
0, // 0x9c 156
0, // 0x9d 157
0, // 0x9e 158
0, // 0x9f 159

sc_SPACE, // [00A0] NO-BREAK_SPACE
sc_SYMBOL, // [00A1] INVERTED_EXCLAMATION_MARK
sc_SYMBOL, // [00A2] CENT_SIGN
sc_SYMBOL, // [00A3] POUND_SIGN
sc_SYMBOL, // [00A4] CURRENCY_SIGN
sc_SYMBOL, // [00A5] YEN_SIGN
sc_SYMBOL, // [00A6] BROKEN_BAR
sc_SYMBOL, // [00A7] SECTION_SIGN
sc_SYMBOL, // [00A8] DIAERESIS
sc_SYMBOL, // [00A9] COPYRIGHT_SIGN
sc_SYMBOL, // [00AA] FEMININE_ORDINAL_INDICATOR
sc_SYMBOL, // [00AB] LEFT-POINTING_DOUBLE_ANGLE_QUOTATION_MARK
sc_SYMBOL, // [00AC] NOT_SIGN
sc_SYMBOL, // [00AD] SOFT_HYPHEN
sc_SYMBOL, // [00AE] REGISTERED_SIGN
sc_SYMBOL, // [00AF] MACRON
sc_SYMBOL, // [00B0] DEGREE_SIGN
sc_SYMBOL, // [00B1] PLUS-MINUS_SIGN
sc_SYMBOL, // [00B2] SUPERScript_TWO
sc_SYMBOL, // [00B3] SUPERScript_THREE
sc_SYMBOL, // [00B4] ACUTE_ACCENT
sc_SYMBOL, // [00B5] MICRO_SIGN
sc_SYMBOL, // [00B6] PILCROW_SIGN
sc_SYMBOL, // [00B7] MIDDLE_DOT
sc_SYMBOL, // [00B8] CEDILLA
sc_SYMBOL, // [00B9] SUPERScript_ONE
sc_SYMBOL, // [00BA] MASCULINE_ORDINAL_INDICATOR
sc_SYMBOL, // [00BB] RIGHT-POINTING_DOUBLE_ANGLE_QUOTATION_MARK
sc_SYMBOL, // [00BC] VULGAR_FRACTION_ONE_QUARTER

```

```

sc_SYMBOL, // [00BD] VULGAR_FRACTION_ONE_HALF
sc_SYMBOL, // [00BE] VULGAR_FRACTION_THREE_QUARTERS
sc_SYMBOL, // [00BF] INVERTED_QUESTION_MARK

sc_ALPHA|sc_UPCASE, // [00C0] LATIN_CAPITAL_LETTER_A_WITH_GRAVE
sc_ALPHA|sc_UPCASE, // [00C1] LATIN_CAPITAL_LETTER_A_WITH_ACUTE
sc_ALPHA|sc_UPCASE, // [00C2] LATIN_CAPITAL_LETTER_A_WITH_CIRCUMFLEX
sc_ALPHA|sc_UPCASE, // [00C3] LATIN_CAPITAL_LETTER_A_WITH_TILDE
sc_ALPHA|sc_UPCASE, // [00C4] LATIN_CAPITAL_LETTER_A_WITH_DIAERESIS
sc_ALPHA|sc_UPCASE, // [00C5] LATIN_CAPITAL_LETTER_A_WITH_RING_ABOVE
sc_ALPHA|sc_UPCASE, // [00C6] LATIN_CAPITAL_LIGATURE_AE
sc_ALPHA|sc_UPCASE, // [00C7] LATIN_CAPITAL_LETTER_C_WITH_CEDILLA
sc_ALPHA|sc_UPCASE, // [00C8] LATIN_CAPITAL_LETTER_E_WITH_GRAVE
sc_ALPHA|sc_UPCASE, // [00C9] LATIN_CAPITAL_LETTER_E_WITH_ACUTE
sc_ALPHA|sc_UPCASE, // [00CA] LATIN_CAPITAL_LETTER_E_WITH_CIRCUMFLEX
sc_ALPHA|sc_UPCASE, // [00CB] LATIN_CAPITAL_LETTER_E_WITH_DIAERESIS
sc_ALPHA|sc_UPCASE, // [00CC] LATIN_CAPITAL_LETTER_I_WITH_GRAVE
sc_ALPHA|sc_UPCASE, // [00CD] LATIN_CAPITAL_LETTER_I_WITH_ACUTE
sc_ALPHA|sc_UPCASE, // [00CE] LATIN_CAPITAL_LETTER_I_WITH_CIRCUMFLEX
sc_ALPHA|sc_UPCASE, // [00CF] LATIN_CAPITAL_LETTER_I_WITH_DIAERESIS
sc_ALPHA|sc_UPCASE, // [00D0] LATIN_CAPITAL_LETTER_ETH_(Icelandic)
sc_ALPHA|sc_UPCASE, // [00D1] LATIN_CAPITAL_LETTER_N_WITH_TILDE
sc_ALPHA|sc_UPCASE, // [00D2] LATIN_CAPITAL_LETTER_O_WITH_GRAVE
sc_ALPHA|sc_UPCASE, // [00D3] LATIN_CAPITAL_LETTER_O_WITH_ACUTE
sc_ALPHA|sc_UPCASE, // [00D4] LATIN_CAPITAL_LETTER_O_WITH_CIRCUMFLEX
sc_ALPHA|sc_UPCASE, // [00D5] LATIN_CAPITAL_LETTER_O_WITH_TILDE
sc_ALPHA|sc_UPCASE, // [00D6] LATIN_CAPITAL_LETTER_O_WITH_DIAERESIS

sc_SYMBOL, // [00D7] MULTIPLICATION_SIGN

sc_ALPHA|sc_UPCASE, // [00D8] LATIN_CAPITAL_LETTER_O_WITH_STROKE
sc_ALPHA|sc_UPCASE, // [00D9] LATIN_CAPITAL_LETTER_U_WITH_GRAVE
sc_ALPHA|sc_UPCASE, // [00DA] LATIN_CAPITAL_LETTER_U_WITH_ACUTE
sc_ALPHA|sc_UPCASE, // [00DB] LATIN_CAPITAL_LETTER_U_WITH_CIRCUMFLEX
sc_ALPHA|sc_UPCASE, // [00DC] LATIN_CAPITAL_LETTER_U_WITH_DIAERESIS
sc_ALPHA|sc_UPCASE, // [00DD] LATIN_CAPITAL_LETTER_Y_WITH_ACUTE
sc_ALPHA|sc_UPCASE, // [00DE] LATIN_CAPITAL_LETTER_THORN_(Icelandic)

sc_ALPHA|sc_LOCASE, // [00DF] LATIN_SMALL_LETTER_SHARP_S_(German)

sc_ALPHA|sc_LOCASE, // [00E0] LATIN_SMALL_LETTER_A_WITH_GRAVE
sc_ALPHA|sc_LOCASE, // [00E1] LATIN_SMALL_LETTER_A_WITH_ACUTE
sc_ALPHA|sc_LOCASE, // [00E2] LATIN_SMALL_LETTER_A_WITH_CIRCUMFLEX
sc_ALPHA|sc_LOCASE, // [00E3] LATIN_SMALL_LETTER_A_WITH_TILDE
sc_ALPHA|sc_LOCASE, // [00E4] LATIN_SMALL_LETTER_A_WITH_DIAERESIS
sc_ALPHA|sc_LOCASE, // [00E5] LATIN_SMALL_LETTER_A_WITH_RING_ABOVE
sc_ALPHA|sc_LOCASE, // [00E6] LATIN_SMALL_LIGATURE_AE
sc_ALPHA|sc_LOCASE, // [00E7] LATIN_SMALL_LETTER_C_WITH_CEDILLA
sc_ALPHA|sc_LOCASE, // [00E8] LATIN_SMALL_LETTER_E_WITH_GRAVE
sc_ALPHA|sc_LOCASE, // [00E9] LATIN_SMALL_LETTER_E_WITH_ACUTE
sc_ALPHA|sc_LOCASE, // [00EA] LATIN_SMALL_LETTER_E_WITH_CIRCUMFLEX
sc_ALPHA|sc_LOCASE, // [00EB] LATIN_SMALL_LETTER_E_WITH_DIAERESIS
sc_ALPHA|sc_LOCASE, // [00EC] LATIN_SMALL_LETTER_I_WITH_GRAVE
sc_ALPHA|sc_LOCASE, // [00ED] LATIN_SMALL_LETTER_I_WITH_ACUTE
sc_ALPHA|sc_LOCASE, // [00EE] LATIN_SMALL_LETTER_I_WITH_CIRCUMFLEX
sc_ALPHA|sc_LOCASE, // [00EF] LATIN_SMALL_LETTER_I_WITH_DIAERESIS
sc_ALPHA|sc_LOCASE, // [00F0] LATIN_SMALL_LETTER_ETH_(Icelandic)
sc_ALPHA|sc_LOCASE, // [00F1] LATIN_SMALL_LETTER_N_WITH_TILDE
sc_ALPHA|sc_LOCASE, // [00F2] LATIN_SMALL_LETTER_O_WITH_GRAVE
sc_ALPHA|sc_LOCASE, // [00F3] LATIN_SMALL_LETTER_O_WITH_ACUTE
sc_ALPHA|sc_LOCASE, // [00F4] LATIN_SMALL_LETTER_O_WITH_CIRCUMFLEX
sc_ALPHA|sc_LOCASE, // [00F5] LATIN_SMALL_LETTER_O_WITH_TILDE
sc_ALPHA|sc_LOCASE, // [00F6] LATIN_SMALL_LETTER_O_WITH_DIAERESIS

sc_SYMBOL, // [00F7] DIVISION_SIGN

sc_ALPHA|sc_LOCASE, // [00F8] LATIN_SMALL_LETTER_O_WITH_STROKE
sc_ALPHA|sc_LOCASE, // [00F9] LATIN_SMALL_LETTER_U_WITH_GRAVE
sc_ALPHA|sc_LOCASE, // [00FA] LATIN_SMALL_LETTER_U_WITH_ACUTE
sc_ALPHA|sc_LOCASE, // [00FB] LATIN_SMALL_LETTER_U_WITH_CIRCUMFLEX
sc_ALPHA|sc_LOCASE, // [00FC] LATIN_SMALL_LETTER_U_WITH_DIAERESIS
sc_ALPHA|sc_LOCASE, // [00FD] LATIN_SMALL_LETTER_Y_WITH_ACUTE
sc_ALPHA|sc_LOCASE, // [00FE] LATIN_SMALL_LETTER_THORN_(Icelandic)

```

File: SCCSPECL.C

\$Header: /Projects/Toolbox/ct/SCCSPECL.CPP 2 5/30/97 8:45a Wmanis \$

Contains: Maintains typespec list.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```
*****/
#include "scpubobj.h"
```

```
/* ===== */
```

```
void scTypeSpecList::Insert( TypeSpec& ts )
```

```
{
    for ( int i = 0; i < NumItems(); i++ ) {
        if ( ts.ptr() == (*this)[i].ptr() )
            return;
    }
    Append( ts );
}
```

```
/* ===== */
```

```
scKeyRecord::scKeyRecord() :
```

```
{
    type_( insert ),
    fKeyCode( 0 ),
    fReplacedChar( 0 ),
    field_( 0 ),
    replacedfield_( 0 ),
    fEscapement( 0 ),
    fSpec( 0 ),
    fNoOp( 0 ),
    fRestoreSelect( 0 )
}
```

```
/* ===== */
```

```
scKeyRecord::scKeyRecord( const scKeyRecord& rec )
```

```
{
    type_      = rec.type_;
    fKeyCode   = rec.fKeyCode;
    field_     = rec.field_;
    fReplacedChar = rec.fReplacedChar;
    replacedfield_ = rec.replacedfield_;
    fEscapement = rec.fEscapement;
    fSpec      = rec.fSpec;
    fNoOp      = rec.fNoOp;
    fRestoreSelect = rec.fRestoreSelect;
    fMark      = rec.fMark;
}
```

```
/* ===== */
```

```
scKeyRecord& scKeyRecord::operator=( const scKeyRecord& rec )
```

```
{
    type_      = rec.type_;
}
```

```

    fKeyCode      = rec.fKeyCode;
    field_        = rec.field_;
    fReplacedChar  = rec.fReplacedChar;
    replacedfield_ = rec.replacedfield_;
    fEscapement    = rec.fEscapement;
    fSpec          = rec.fSpec;
    fNoOp          = rec.fNoOp;
    fRestoreSelect = rec.fRestoreSelect;
    fMark          = rec.fMark;

    return *this;
}

/* ===== */

scKeyRecord::~scKeyRecord()
{
}

/* ===== */

void scKeyRecord::Invert()
{
    UCS2 tmpChar = fReplacedChar;
    fReplacedChar = fKeyCode;
    fKeyCode = tmpChar;

    uint8 tmpfield = replacedfield_;
    replacedfield_ = field_;
    field_ = tmpfield;
}

/* ===== */

scStreamLocation& scStreamLocation::operator=( const scStreamLocation& sl )
{
    fStream      = sl.fStream;
    fAPPColumn    = sl.fAPPColumn;
    fParaNum      = sl.fParaNum;
    fParaOffset   = sl.fParaOffset;
    fEndOfLine    = sl.fEndOfLine;
    fTheCh        = sl.fTheCh;
    fFlags        = sl.fFlags;
    fUnitType     = sl.fUnitType;
    fTheChWidth   = sl.fTheChWidth;
    fChSpec       = sl.fChSpec;
    fParaSpec     = sl.fParaSpec;
    fPosOnLine    = sl.fPosOnLine;
    fSelMaxX      = sl.fSelMaxX;
    fFont         = sl.fFont;
    fPointSize    = sl.fPointSize;
    fBaseline     = sl.fBaseline;
    fMeasure      = sl.fMeasure;
    fLetterSpace  = sl.fLetterSpace;
    fWordSpace    = sl.fWordSpace;

    return *this;
}

/* ===== */

scStreamLocation::scStreamLocation( const scStreamLocation& sl ) :
    fStream( sl.fStream ),
    fAPPColumn( sl.fAPPColumn ),
    fParaNum( sl.fParaNum ),
    fParaOffset( sl.fParaOffset ),
    fEndOfLine( sl.fEndOfLine ),
    fTheCh( sl.fTheCh ),
    fFlags( sl.fFlags ),
    fUnitType( sl.fUnitType ),
    fTheChWidth( sl.fTheChWidth ),
    fChSpec( sl.fChSpec ),
    fParaSpec( sl.fParaSpec ),

```



```

    fPosOnLine( sl.fPosOnLine ),
    fSelMaxX( sl.fSelMaxX ),
    fFont( sl.fFont ),
    fPointSize( sl.fPointSize ),
    fBaseline( sl.fBaseline ),
    fMeasure( sl.fMeasure ),
    fLetterSpace( sl.fLetterSpace ),
    fWordSpace( sl.fWordSpace )
{
}

```

```
/* ===== */
```

```
scStreamLocation::scStreamLocation() :
```

```

    fStream( 0 ),
    fAPPColumn( 0 ),
    fParaNum( 0 ),
    fParaOffset( 0 ),
    fEndOfLine( 0 ),
    fTheCh( 0 ),
    fFlags( 0 ),
    fUnitType( eNoUnit ),
    fTheChWidth( 0 ),
    fChSpec( 0 ),
    fParaSpec( 0 ),
    fPosOnLine( 0 ),
    fSelMaxX( 0 ),
    fFont( 0 ),
    fPointSize( 0 ),
    fBaseline( 0 ),
    fMeasure( 0 ),
    fLetterSpace( 0 ),
    fWordSpace( 0 )

```

```
/* ===== */
```

```
void scStreamLocation::Init()
```

```

    fStream          = 0;
    fAPPColumn       = 0;
    fParaNum         = 0;
    fParaOffset      = 0;
    fEndOfLine       = 0;
    fTheCh           = 0;
    fFlags           = 0;
    fUnitType        = eNoUnit;
    fTheChWidth      = 0;
    fChSpec.clear();
    fParaSpec.clear();
    fPosOnLine       = 0;
    fSelMaxX         = 0;
    fFont            = 0;
    fPointSize       = 0;
    fBaseline        = 0;
    fMeasure         = 0;
    fLetterSpace     = 0;
    fWordSpace       = 0;
}

```

```
/* ===== */
```

```
#if SCDEBUG > 1
```

```
void scSpecLocList::DbgPrint( void ) const
```

```

{
    SCDebugTrace( 0, scString( "\nSCSPECLOCLIST\n" ) );
    for ( int i = 0; i < NumItems(); i++ ) {
        SCDebugTrace( 0, scString( "\tscCharSpecLoc ( %d %d ) 0x%08x\n",
            (*this)[i].offset().fParaOffset,
            (*this)[i].offset().fCharOffset,
            (*this)[i].spec() );
    }
}

```

```

    SCDebugTrace( 0, scString( "SCSPECLOCLIST\n" ) );
}
#endif

```

```

/* ===== */

```

```

TypeSpec scSpecLocList::GetLastValidSpec( void ) const
{
    for ( int i = NumItems() - 1; i >= 0; i-- ) {
        if ( (*this)[i].spec().ptr() )
            return (*this)[i].spec();
    }
    return 0;
}

```

```

/* ===== */

```

```

TypeSpec scSpecLocList::GetFirstValidSpec( void ) const
{
    for ( int i = 0; i < NumItems(); i++ ) {
        if ( (*this)[i].spec().ptr() )
            return (*this)[i].spec();
    }
    return 0;
}

```

```

/* ===== */

```

```

TypeSpec scSpecLocList::GetNthValidSpec( int nth ) const
{
    for ( int i = 0; i < NumItems(); i++ ) {
        if ( (*this)[i].spec().ptr() && --nth == 0 )
            return (*this)[i].spec();
    }
    return 0;
}

```

```

/* ===== */

```

```

    sc_ALPHA|sc_LOCASE,          // [00FF] LATIN_SMALL_LETTER_Y_WITH_DIAERESIS
    0
};

/* ===== */

static UCS2 CTChangeCase( UCS2 );

/* ===== */
/* return the lower case of a character */

UCS2 CTTToLower( UCS2 ch )
{
    register ushort test;

    if ( ch < 256 ) {
        test = sc_CharType[ch+1];
        if ( test & sc_UPCASE ) {
            if ( ch != 0xDF )
                return (UCS2)(ch + 0x20);
        }
    }
    else
        ; /* case may not be significant */

    return ch;
}

/* ===== */

UCS2 CTTToUpper( register UCS2 ch )
{
    register ushort test;

    if ( ch < 256 ) {
        test = sc_CharType[ch+1];
        if ( test & sc_LOCASE ) {
            if ( ch != 0xFF )
                return (UCS2)(ch - 0x20);
        }
    }
    else
        ; /* case may not be significant */

    return ch;
}

/* ===== */

UCS2 CTToggleCase( register UCS2 ch )
{
    register ushort test;

    if ( ch < 256 ) {
        test = sc_CharType[ch+1];
        if ( test & sc_LOCASE ) {
            if ( ch != 0xFF )
                return (UCS2)(ch - 0x20);
        }
        else if ( test & sc_UPCASE ) {
            if ( ch != 0xDF )
                return (UCS2)(ch + 0x20);
        }
    }
    else
        ; /* case may not be significant */

    return ch;
}

```

[illegible]

File: SCCOLUMN.H

\$Header: /Projects/Toolbox/ct/SCCOLUMN.H 2 5/30/97 8:45a Wmanis \$

Contains: text container definitions

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```
*****/
#ifndef _H_SCCOLUMN
#define _H_SCCOLUMN

#include "sctbobj.h"
```

```

===== */
===== */
===== */
```

```
class scRedisplList;
class scSelection;
class TextMarker;
class scImmediateRedispl;
class stTextImportExport;
class scLINERefData;
class scCOLRefData;
class scXRect;
class scRedisplList;
class scTypeSpecList;
class scLineInfoList;
class scSpecLocList;
/* THE COLUMN OBJECT */
```

```
class scColumn : public scTBObj {
    scDECLARE_RTTI;
```

```
friend class scCOLRefData;
```

```
public:
```

```
    scColumn( APPColumn,
              MicroPoint,
              MicroPoint,
              scStream* p      = 0,
              eCommonFlow flow = eRomanFlow );
```

```
    scColumn() :
        fShapePieces( 0 ),
        fRgnH( 0 ),
        fNextContext( 0 ),
        fAppName( 0 ),
        fColumnCount( 0 ),
        fSize( 0, 0 ),
        fFlowDir( eRomanFlow ),
        fStream( 0 ),
        fSelection( 0 ),
        fFirstline( 0 ){}

    ~scColumn();
```

```

////////////////////////////////////
// FLOW SET METHODS //
////////////////////////////////////

```

```

        // delete the stream from the flowset
        // RETURNS the damaged area(s)
        //
void      FlowsetClearStream( scRedisplList* );

        // remove the stream from the flowset
        // RETURNS the damaged area(s)
        //
void      FlowsetCutStream( scStream*, scRedisplList*);

        // paste the stream into the flowset
        // RETURNS the damaged area(s)
        //
void      FlowsetPasteStream( scStream*, scRedisplList* );

        // get the selection object associated with
        // the flowset, if there is none it will
        // create one
        //
scSelection* FlowsetGetSelection( void );

        // set the selection object for the flowset
        // none should exist, since if it does
        // error recovery might be a bit tricky
        //
void      FlowsetSetSelection( scSelection* );

        // this removes the selection from the flowset
        // NOTE: it does not delete it
        //
void      FlowsetRemoveSelection( void );

        // invalidate any selection associated with
        // the flowset
void      FlowsetInvalidateSelection( void );

        // set the flow for the flowset
        // all containers in a flowset must have the
        // same flow at this time
void      FlowsetSetFlowdir( const scFlowDir& );

scColumn* GetFlowset( void ) const
{
    return (scColumn*)FirstInChain();
}

void      RecomposeFlowset( long          ticks          = LONG_MAX,
                           scRedisplList* redispllist    = 0 );

```

```

////////////////////////////////////
// COLUMN METHODS //
////////////////////////////////////

```

```

void      Enumerate( long& );

        // draw the column updating the area
        // intersected by the damage rect
        //
virtual void Draw( const scXRect& damagedRect,

```

```
APPCTX,
const scMuPoint* translation = 0 );
```

```
void Hilite( const TextMarker&,
             const TextMarker&,
             HiliteFuncPtr,
             const scSelection& selection );
```

```
// FILE I/O
```

```
// complete the read
virtual void Read( scSet*, APPCTXPtr, IOFuncPtr );
```

```
// complete the write
virtual void Write( APPCTXPtr, IOFuncPtr );
```

```
// restore the pointers after completing a read
virtual void RestorePointers( scSet* );
```

```
void SetRecomposition( Bool tf );
Bool GetRecomposition( void ) const;
```

```
// get or set the first line of the column
//
```

```
scTextline* GetFirstline( void ) const
{
    return fFirstline;
}
```

```
void SetFirstline( scTextline* txl )
{
    fFirstline = txl;
}
```

```
scTextline* GetLastline( void ) const;
```

```
void TranslateLines( const scMuPoint& );
void RepositionLines( void );
```

```
scContUnit* MarkParas( void );
scContUnit* LastPara( void ) const;
```

```
// return the first paragraph in this container
// for reformatting purposes, we will assume that
// the previous container has been successfully
// reformatted
```

```
scContUnit* FirstPara( void ) const;
```

```
// return the number of lines for this column,
// if it is not formatted it will return -1
```

```
ushort GetLinecount( void ) const;
```

```
virtual void Resize( const scSize& size, scRedisplList* = 0 );
void Resize( MicroPoint, MicroPoint, scRedisplList* = 0 );
```

```
scXRect& RepaintExtent( scXRect& );
void QueryMargins( scXRect& ) const;
void QuerySize( scSize& ) const;
void QueryTextDepth( MicroPoint& ) const;
MicroPoint TextDepth( void ) const;
void GetTSList( scTypeSpecList& ) const;
```

```
// should we reformat this column or wait till later
Bool DamOpen( void );
```

```

void      Rebreak( scRedisplist* );
void      Rebreak2( scRedisplist* );

void      ExternalSize( long& );
void      ZeroEnumeration( void );

Bool      GetStrip( scLINERefData&, int, scCOLRefData& );

void      DeleteExcessLines( scContUnit*, scTextline*, Bool, scCOLRefData& );

```

```

////////////////////////////////////
// COLUMN SHAPE METHODS //
////////////////////////////////////

```

```

void      ReplacePoly( scVerthandle, scRedisplist* );
void      PastePoly( scVerthandle, scRedisplist* );
void      CopyPoly( scVerthandle* );

void      PasteRgn( const HRgnHandle, scRedisplist* );
void      CopyRgn( HRgnHandle& );

void      ClearShape( scRedisplist* );

```

```

////////////////////////////////////
// COLUMN LINKAGE METHODS //
////////////////////////////////////

```

```

void      Link( scColumn*, Bool, scRedisplist* );
void      Unlink( scRedisplist* );

void      Renumber( void );

void      BreakChain( scColumn* );

        // get the next or previous column that
        // actually contains lines (i.e. composed text )
        //
scColumn* PrevWithLines( void ) const;
scColumn* NextWithLines( void ) const;

void      ComputeInkExtents( void );

void      SetInkExtents( MicroPoint x1, MicroPoint y1, MicroPoint x2, MicroPoint y2 )
        {
            fInkExtents.Set( x1, y1, x2, y2 );
        }

const scXRect& GetInkExtents( void ) const
        {
            return fInkExtents;
        }

void      UnionInkExtents( const scXRect& xrect )
        {
            fInkExtents.Union( xrect );
        }

Bool      MoreText( void ) const;
Bool      HasText( void ) const;

scStream* GetStream( void ) const
        {
            return fStream;
        }

void      SetStream( scStream* stream )
        {
            fStream = stream;
        }

void      SetFlowsetStream( scStream* stream );
void      FreeStream( void );

```



```

eVertJust      GetVertJust( void ) const
                {
                    return (eVertJust)fLayBits.fLayAdjustment;
                }
void            SetVertJust( eVertJust vj )
                {
                    fLayBits.fLayAdjustment = vj;
                }

eColShapeType  GetShapeType( void ) const
                {
                    return (eColShapeType)fLayBits.fLayType;
                }
void            SetShapeType( eColShapeType st );

ushort         GetShapePieces( void ) const
                {
                    return fShapePieces;
                }

scVertHandle    GetVertList( void ) const
                {
                    return fVertH;
                }
void            SetVertList( scVertHandle vl )
                {
                    fVertH = vl;
                }

HRgnHandle      GetRgn( void ) const
                {
                    return fRgnH;
                }
void            SetRgn( HRgnHandle rgn )
                {
                    fRgnH = rgn;
                }

void            SetAPPName( APPColumn appcol )
                {
                    fAppName = appcol;
                }
APPColumn       GetAPPName( void ) const
                {
                    return fAppName;
                }

void            SetWidth( MicroPoint w )
                {
                    fSize.SetWidth( w );
                }
MicroPoint      Width( void ) const
                {
                    return fSize.Width();
                }

void            SetDepth( MicroPoint d )      { fSize.SetDepth( d ); }
MicroPoint      Depth( void ) const           { return fSize.Depth(); }

void            SetSize( const scSize& size )
                {
                    fSize = size;
                }
const scSize&    GetSize( void ) const
                {
                    return fSize;
                }

void            SetSize( MicroPoint w, MicroPoint d )
                {
                    fSize.SetWidth( w ), fSize.SetDepth( d );
                }

```

```

    }

    scColumn*      GetPrev( void ) const
    {
        return (scColumn*)Prev();
    }

    scColumn*      GetNext( void ) const          { return (scColumn*)Next(); }

    void           SetCount( long count )
    {
        fColumnCount = count;
    }

    long           GetCount( void ) const
    {
        return fColumnCount;
    }

    void           SetFlowdir( const scFlowDir& fd )
    {
        fFlowDir = fd;
    }

    const scFlowDir& GetFlowdir( void ) const
    {
        return fFlowDir;
    }

    void           SetContext( scColumn* ctx )
    {
        fNextContext = ctx;
    }

    scColumn*      GetContext( void ) const
    {
        return fNextContext;
    }

    void           SetVertFlex( Bool, scRedisplList* );
    void           SetHorzFlex( Bool, scRedisplList* );
    Bool           GetVertFlex( void ) const
    {
        return GetShapeType() & eVertFlex;
    }

    Bool           GetHorzFlex( void ) const
    {
        return GetShapeType() & eHorzFlex;
    }

    void           Delete( scRedisplList* );
    void           Free( void );
    void           FreeShape( void );
    void           FreeScrap( void );
    void           UpdateLine( scImmediateRedispl&, APPDrwCtx );

    void           LineExtents( scImmediateRedispl& );
    void           FreeLines( Bool, scXRect& );

    void           InvertExtents( HiliteFuncPtr, APPDrwCtx );
#if SCDEBUG > 1
    virtual void    scAssertValid( Bool recurse = true ) const;
    void           DbgPrintInfo( int debugLevel = 0 ) const;
#else
    virtual void    scAssertValid( Bool = true ) const{}
#endif
    #endif

    static scColumn* FindFlowset( const scStream* );

    // context list
    static scColumn* GetBaseContextList( void )
    {
        return fTheContextList;
    }

    static void      FiniCTXList( void );

```

```

void      AddToCTXList( )
        {
            fNextContext    = fTheContextList;
            fTheContextList = this;
        }
void      DeleteFromCTXList( );
void      VerifyCTXList( void ) const;

static void ChangedTS( TypeSpec, eSpecTask, scRedisplList* );
static void Update( scRedisplList* );

void      LineInfo( scLineInfoList*,
                    long&,
                    Bool& ) const;

void      VertJustify( void );
void      SetDepthNVJ( MicroPoint, scRedisplList* );
void      SetVJ( eVertJust );

// COLUMN SELECTION
void      ClickEvaluate( const scMuPoint&,
                        REAL& );

void      StartShiftClick( scStreamLocation&,
                           const scMuPoint&,
                           HiliteFuncPtr,
                           APPDrwCtx,
                           scSelection*& );

void      StartClick( const scMuPoint&,
                      HiliteFuncPtr,
                      APPDrwCtx,
                      scSelection*& );

void      ContinueClick( const scMuPoint&,
                         HiliteFuncPtr,
                         scSelection* );

Bool      Select( const scMuPoint&    hitPt,
                  TextMarker*         textMarker,
                  REAL*                bestDist );

void      InitialSelection( TypeSpec&, scSelection*& );

void      SelectSpecial( const scMuPoint&,
                         eSelectModifier,
                         scSelection*& );

void      LimitDamage( scRedisplList*, long );

void      PasteAPPText( stTextImportExport&, scRedisplList* );

void      ReadTextFile( TypeSpec,
                        APPCtxPtr,
                        IOFuncPtr,
                        scRedisplList* );

protected:

        // do not confuse the following with flowset operations
scSelection* GetSelection( void )
        {
            return fSelection;
        }

```

```

    }
void      SetSelection( scSelection* sel )
    {
        fSelection = sel;
    }

    // actually allocate the real estate for lines
virtual Bool GetStrip2( scLINERefData&, int, scCOLRefData& );

private:

    static scColumn*      fTheContextList;

    void      CreateSelection( void );

    scColumn*      fNextContext;

    APPColumn      fAppName;          // application name

    long           fColumnCount;

#ifdef 0
    MicroPoint     fWidth;            // width of column
    MicroPoint     fDepth;           // depth of column
#else
    scSize         fSize;
#endif

    scFlowDir      fFlowDir;          // the basic flow direction of a container

    scStream*      fStream;           // hook into stream
    scSelection*   fSelection;

    scXRect        fInkExtents;       // actual extents w/ italics, idents, etc.

    scTextline*    fFirstline;        // firstline of the column

    ushort         fShapePieces;      // num of components of shape
    union {
        scVertHandle fVerth;
        HRgnHandle   fRgnH;
    };
};

/*-----*/

#define FIRST_LINE_POSITION      (LONG_MIN + 1)
#define HorzFlexMeasure         (LONG_MAX - one_pica)

/* these seems arbitrary,
 * but we need to get it
 * away from LONG_MAX
 */

/* ----- */

/* OPTIMIZATIONS */
#define COLShapePieces( c ) ( (c)->fShapePieces )

/* PROTOTYPES */
/*****
/*****
short      COLLineNum( scSelection* );

/*****
/*****

```


File: SCCOLUMN.C

\$Header: /Projects/Toolbox/ct/Sccolumn.cpp 4 5/30/97 8:45a Wmanis \$

Contains: The 'methods' for the column objects.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/
#include "sccolumn.h"
#include "scpubobj.h"
#include "scaptex.h"
#include "sccallbk.h"
#include "scstcach.h"
#include "scglobda.h"
#include "scmem.h"
#include "scparagr.h"
#include "scpolygo.h"
#include "scregion.h"
#include "scselect.h"
#include "scstream.h"
#include "scset.h"
#include "sctextli.h"
#include "scrfdat.h"
#include "scfileio.h"
#include <float.h>
/*
***** */
/*
***** */
scColumn* scColumn::fTheContextList = 0;
/*
***** */
scColumn::~scColumn()
{
    delete fSelection, fSelection = 0;
}
/*
***** */
#define FILE_SIZE_COLUMN    28

void scColumn::Read( scSet*      enumTable,
                    APPCtxPtr  ctxPtr,
                    IOFuncPtr  readFunc )
{
    uchar          abuf[FILE_SIZE_COLUMN];
    const uchar*   pbuf    = abuf;

    scTBObj::Read( enumTable, ctxPtr, readFunc );
    Mark( scINVALID );

    // read in the rest of the columns data
    ReadBytes( abuf, ctxPtr, readFunc, FILE_SIZE_COLUMN );

    // pointer to stream
    along uval;

```

```

pbuf = BufGet_long( pbuf, uval, kIntelOrder );
fStream = (scStream*)uval;

    // pointer to first line
pbuf = BufGet_long( pbuf, uval, kIntelOrder );
scAssert( uval == 0 );

    // flow direction
ushort uflow;
pbuf = BufGet_short( pbuf, uflow, kIntelOrder );
fFlowDir.SetLineDir( (eTextDirections)uflow );

pbuf = BufGet_short( pbuf, uflow, kIntelOrder );
fFlowDir.SetCharDir( (eTextDirections)uflow );

    // width & depth
pbuf = BufGet_long( pbuf, uval, kIntelOrder );
fSize.SetWidth( uval );

pbuf = BufGet_long( pbuf, uval, kIntelOrder );
fSize.SetDepth( uval );

    // application name
pbuf = BufGet_long( pbuf, uval, kIntelOrder );
fAppName = (APPColumn)APPDiskIDToPointer( ctxPtr, (long)uval, diskidColumn );

    // count
pbuf = BufGet_long( pbuf, uval, kIntelOrder );
fColumnCount = uval;

scAssert ( (size_t)(pbuf-abuf) == FILE_SIZE_COLUMN );

    // shape type
long val;
ReadLong( val, ctxPtr, readFunc, kIntelOrder );

if ( val ) {
    HRgnHandle rgnH = RGNfromFile( ctxPtr, readFunc, fShapePieces );

    SetShapeType( eRgnShape );

    fRgnH = rgnH;

    scAutoUnlock h( fRgnH );
    HRgn* rgn = (HRgn *)h;

    fShapePieces = (ushort)rgn->fNumSlivers;
}
else
    fShapePieces = 0;

if ( !GetPrev() )
    scStream::STRFromFile( enumTable, ctxPtr, readFunc );
}

/* ===== */
/* ACTUAL WRITE, this performs the write out of the column data structure,
 * paragraphs are written out with the first column in a set of linked columns
 * other than the column itself the only thing we will be writting out will
 * be the outline vertices
 */

void scColumn::Write( APPCtxPtr ctxPtr,
                    IOFuncPtr writeFunc )
{
    scTBObj::Write( ctxPtr, writeFunc );

    uchar abuf[FILE_SIZE_COLUMN];
    uchar* pbuf = abuf;

    // pointer to stream
pbuf = BufSet_long( pbuf, fStream ? fStream->GetEnumCount() : 0, kIntelOrder );

```

```

    // pointer to first line
    pbuf = BufSet_long( pbuf, 0, kIntelOrder );

    // flow direction
    pbuf = BufSet_short( pbuf, (ushort)fFlowDir.GetLineDir(), kIntelOrder );
    pbuf = BufSet_short( pbuf, (ushort)fFlowDir.GetCharDir(), kIntelOrder );

    // width & depth
    pbuf = BufSet_long( pbuf, fSize.Width(), kIntelOrder );
    pbuf = BufSet_long( pbuf, fSize.Depth(), kIntelOrder );

    // application name
    pbuf = BufSet_long( pbuf,
        APPPointerToDiskID( ctxPtr, fAppName, diskidColumn ),
        kIntelOrder );

    // count
    pbuf = BufSet_long( pbuf, GetCount(), kIntelOrder );

    scAssert ((size_t)(pbuf-abuf) == FILE_SIZE_COLUMN );

    WriteBytes( abuf, ctxPtr, writeFunc, FILE_SIZE_COLUMN );

    WriteLong( (ulong)fShapePieces, ctxPtr, writeFunc, kIntelOrder );

    switch ( GetShapeType() ) {
        default:
            break;

        case eVertShape:
            POLYtoFile( ctxPtr, writeFunc, fVertH, fShapePieces );
            break;

        case eRgnShape:
            RGNtoFile( ctxPtr, writeFunc, fRgnH, fShapePieces );
            break;
    }

    if ( !GetPrev() )
        fStream->STRtoFile( ctxPtr, writeFunc );

    /* ===== */
void scColumn::RestorePointers( scSet* enumTable )
{
    if ( !Marked( scPTRRESTORED ) ) {
        scTObj::RestorePointers( enumTable );

        AddToCTXList();

        fStream = (scStream*)enumTable->Get( (long)fStream );
        if ( !GetPrev() )
            fStream->STRRestorePointers( enumTable );
    }
}

/* ===== */
void scColumn::SetRecomposition( Bool tf )
{
    scColumn* col = (scColumn*)FirstInChain();

    if ( tf )
        col->Mark( scLAYcomposeACTIVE );
    else
        col->Unmark( scLAYcomposeACTIVE );
}

/* ===== */

```



```

Bool scColumn::GetRecomposition( void ) const
{
    scColumn* col = (scColumn*)FirstInChain();

    return col->Marked( scLAYcomposeACTIVE );
}

/* ===== */
// get the selection object associated with the flowset, if there is
// none it will create one

scSelection* scColumn::FlowsetGetSelection( void )
{
    scColumn* col = (scColumn*)FirstInChain();

    if ( !col->GetSelection() )
        col->SetSelection( SCNEW scSelection( col ) );

    return col->GetSelection();
}

/* ===== */
// set the selection object for the flowset none should exist,
// since if it does error recovery might be a bit tricky

void scColumn::FlowsetSetSelection( scSelection* sel )
{
    scColumn* col = (scColumn*)FirstInChain();
    col->SetSelection( sel );
}

/* ===== */
// this removes the selection from the flowset
// NOTE: it does not delete it

void scColumn::FlowsetRemoveSelection( void )
{
    scColumn* col = (scColumn*)FirstInChain();
    col->SetSelection( 0 );
}

/* ===== */

void scColumn::FlowsetInvalidateSelection( void )
{
    scColumn* col = (scColumn*)FirstInChain();
    scSelection* sel = col->GetSelection();

    if ( sel )
        sel->Invalidate();
}

/* ===== */

void scColumn::RecomposeFlowset( long ticks, scRedisplList* redisplList )
{
    scColumn* col = (scColumn*)FirstInChain();

    SetRecomposition( true );

    for ( ; col; col = col->GetNext() ) {
        if ( col->Marked( scINVALID ) && col->DamOpen() )
            col->LimitDamage( redisplList, ticks );
        else if ( col->Marked( scREALIGN ) ) {
            scRedisplayStoredLine rdl( GetLinecount( ) );
            rdl.SaveLineList( this );

            col->VertJustify();
        }
    }
}

```

```

        scXRect lineDama;
        rdl.LineListChanges( this, lineDamage, redisplList );
        col->Unmark( scREALIGN );
    }
}
scSelection* select = FlowsetGetSelection();
select->UpdateSelection( );
}

/* ===== */
/* ===== */
#ifdef SCDEBUG > 1

void scColumn::scAssertValid( Bool recurse ) const
{
    scTBObj::scAssertValid( recurse );
    if ( !recurse ) {
        if ( fFirstline )
            fFirstline->scAssertValid( false );

        if ( fStream )
            fStream->scAssertValid( false );
    }
}

#endif

/* ===== */
// should we reformat this column or wait till later

Bool scColumn::DamOpen( )
{
    return APPRecomposeColumn( GetAPPName() );
}

/* ===== */
/* set the max selection extent based upon the column flow direction */
static void COLSetSelMax( scColumn*      col,
                          TextMarker*   tm,
                          const scMuPoint& muPt )
{
    if ( col->GetFlowdir().IsVertical() )
        tm->fSelMaxX = muPt.y;
    else
        tm->fSelMaxX = muPt.x;
}

/* ===== */
/* hilite or dehilite the characters in this column */
void scColumn::Hilite( const TextMarker&      tmMark,
                      const TextMarker&      tmPoint,
                      HiliteFuncPtr          func,
                      const scSelection&      selection )
{
    scTextline* txl;
    scTextline* lastTxl;
    scTextline* txl1 = tmMark.fTxl;
    scTextline* txl2 = tmPoint.fTxl;
    MicroPoint startLoc = tmMark.fHLoc;
    MicroPoint stopLoc = tmPoint.fHLoc;
    APPDrwCtx appMat;

    APPDrawContext( GetAPPName(), this, appMat );

    lastTxl = txl2->GetNext();

    for ( txl = txl1; txl && txl != lastTxl; txl = txl->GetNext() ) {
        if ( txl == txl1 )
            txl->Hilite( &tmMark, startLoc, NULL, LONG_MAX, appMat, func, selection );
        else if ( txl == txl2 )
            txl->Hilite( NULL, LONG_MIN, &tmPoint, stopLoc, appMat, func, selection );
    }
}

```

```

    else
        txl->Hilite( NULL, LONG_MIN, NULL, LONG_MAX, appMat, func, selection );
    }
}

/* ===== */
/* select text in a col at the given hit point */

Bool scColumn::Select( const scMuPoint& hitPt,
                      TextMarker*   textMarker,
                      REAL*         bestDist )
{
    scXRect    exRect;
    scTextline* txl;
    long       count;
    scMuPoint   charOrg;
    MicroPoint  fudgeHFactor,
                fudgeVFactor;
    REAL        dist;
    Bool        belowText    = false;
    Bool        vertical     = false;
    int         lineNum;

    vertical = GetFlowdir().IsVertical();

    /* make first hit infinitely far away */
    *bestDist = DBL_MAX;

    textMarker->fCol          = this;
    textMarker->fColCount     = GetCount();
    textMarker->fPara         = NULL;
    textMarker->fTxl         = NULL;

    fudgeHFactor = fudgeVFactor = 0;

    while ( GetFirstline() && !textMarker->fPara ) {
        for ( lineNum = 0, txl = GetFirstline(); txl; txl = LNNext( txl ), lineNum++ ) {

            txl->QueryExtents( exRect );
            // grow hit by fudge factor to account for sloppy hits,
            // how well will this worked on zoomed text?, this value
            // is in world coordinates, NOT the screen coordinates
            exRect.Inset( fudgeHFactor, fudgeVFactor );

            if ( exRect.PinRect( hitPt ) ) {
                SCDEBUG > 1
                SCDebugTrace( 2, scString( "COLSelect: line #%d (%d,%d) (%d %d %d %d)\n" ),
                             lineNum,
                             muPoints( hitPt.x ), muPoints( hitPt.y ),
                             muPoints( exRect.x1 ), muPoints( exRect.y1 ),
                             muPoints( exRect.x2 ), muPoints( exRect.y2 ) );
                #endif

                Bool endOfLine;

                // we have a hit within the extents of the line, now see
                // exactly where on the line we may have selected
                dist = txl->Select( charOrg, count, hitPt, eCursNoMovement, endOfLine );

                if ( dist < *bestDist ) {
                    // we have a hit that is better than any previous hit
                    *bestDist = dist;

                    if ( vertical )
                        textMarker->fHLoc    = charOrg.y;
                    else
                        textMarker->fHLoc    = charOrg.x;

                    textMarker->fOffset     = count;

                    //
                    // if ( LNOrigin( txl ) + LNLength(txl) <= fHLoc && LNIsHyphenated( txl ) )
                    //     textMarker->fEndOfLine = true;
                    // else
                    //     textMarker->fEndOfLine = endOfLine;
                }
            }
        }
    }
}

```

```

        textMarker->fPara      = tx1->GetPara();
        textMarker->fTx1       = tx1;
        textMarker->fParaCount  = textMarker->fPara->GetCount();
        textMarker->fLineCount  = tx1->GetLinecount();
    }
}

/* if no selection and the y position is
 * lower than the top of the last line, then
 * select the last char on the last line
 */

/* assumes lines move from right to left */
if ( vertical )
    belowText = !LNNext( tx1 ) && hitPt.x < exRect.x2 && *bestDist == DBL_MAX;
else
    belowText = !LNNext( tx1 ) && hitPt.y > exRect.y2 && *bestDist == DBL_MAX;

if ( belowText ) {
    *bestDist = tx1->Select( charOrg, count, hitPt, eCursForward, textMarker->fEndOfLine
);

    textMarker->fOffset      = tx1->GetEndOffset();

    scMuPoint charOrg;
    charOrg                 = tx1->Locate( textMarker->fOffset, charOrg, eCursForward );
    if ( vertical )
        textMarker->fHLoc    = charOrg.y;
    else
        textMarker->fHLoc    = charOrg.x;

    textMarker->fPara        = tx1->GetPara();
    textMarker->fTx1         = tx1;
    textMarker->fParaCount   = textMarker->fPara->GetCount();
    textMarker->fLineCount   = tx1->GetLinecount();
    break;
}

}

if ( vertical ) {
    fudgeHFactor -= scPOINTS(1);
    fudgeVFactor -= scPOINTS(8);
}
else {
    fudgeHFactor -= scPOINTS(144);
    fudgeVFactor -= scPOINTS(1);
}
}

return( textMarker->fPara != NULL );
}

/* ===== */
/* return a number that is the square of the dx plus the square of the
 * dy between the 'pt' and a significant point
 */

void scColumn::ClickEvaluate( const scMuPoint& pt,
                             REAL& dist )
{
    TextMarker tm;
    REAL nearDist;

    dist = DBL_MAX;    /* defined in scmath.h */

    if ( GetStream() ) {
        tm.fCol = this;
        raise_if ( !Select( pt, &tm, &nearDist ), scERRlogical );
    }

    dist = nearDist;
}

```

```

/* ===== */
/* select something special indicated by the SelectType */

void scColumn::SelectSpecial( const scMuPoint& pt,
                             eSelectModifier selectMod,
                             scSelection*& select )
{
    select = FlowsetGetSelection();

    scSelection    newSelection( *select );
    REAL          dist;

    if ( !GetStream() )
        return;

    newSelection.fMark.fCol    = this;
    COLSetSelMax( this, &newSelection.fMark, pt );

    if ( selectMod == eAllSelect )
        newSelection.AllSelect();
    else {

#ifdef TESTEXTENTS
        {
            HRect          maxExRect,          /* column extents */
                      maxMargRect;          /* column margins */

            /* if the point is to far out of the maxExRect
             * things will get very slow
             */

            maxExRect = col->fExtents;
            SetHRect( &maxMargRect, 0, 0, col->fWidth, col->fDepth );
            UnionHRect( &maxExRect, &maxMargRect, &maxExRect );

            if ( !MuPtInHRect( pt, &maxExRect ) ) {
                /* the point is in GM's front yard */
                return scERRbounds;
            }
        }
#endif /* TESTEXTENTS */

        raise_if( !Select( pt, &newSelection.fMark, &dist ), scERRbounds );

        newSelection.fPoint    = newSelection.fMark;

        switch ( selectMod ) {
            case eWordSelect:
                newSelection.WordSelect();
                break;
            case eLineSelect:
                newSelection.LineSelect();
                break;
            case eParaSelect:
                newSelection.ParaSelect();
                break;
            case eColumnSelect:
                newSelection.ColumnSelect();
        }
    }

    *select = newSelection;
}

/* ===== */
/* start selection in the original column
 */

```

```

void scColumn::StartClick( const scMuPoint& pt,
                           HiliteFuncPtr   func,
                           APPDrwCtx,
                           scSelection*&    select )
{
    REAL      dist;
    scSelection selection;

    if ( !GetStream() )
        return;

    selection.fMark.fCol = this;
    COLSetSelMax( this, &selection.fMark, pt );
    raise_if ( !Select( pt, &selection.fMark, &dist ), scERRlogical );
    selection.fPoint = selection.fMark;
    selection.LineHilite( func );
    select = FlowsetGetSelection();
    *select = selection;
}

/* ===== */

void scColumn::ContinueClick( const scMuPoint& pt,
                              HiliteFuncPtr   func,
                              scSelection*&    select )
{
    REAL      dist;
    scSelection oldSelection( *select );

    raise_if( !select->fMark.fCol, scERRstructure );

    if ( !GetStream() )
        return;

    select->fPoint.fCol = this;

    if ( !GetFirstline() )
        return;

    // columns not in same stream, application program should catch this
    raise_if ( select->fMark.fCol->GetStream() != select->fPoint.fCol->GetStream(), scERRstructure );

    COLSetSelMax( this, &select->fPoint, pt );

    if ( Select( pt, &select->fPoint, &dist ) ) {
        select->InteractiveHilite( oldSelection, func );
    }
    else
        raise( scERRlogical );
}

/* ===== */

void scColumn::StartShiftClick( scStreamLocation& mark,
                                const scMuPoint& pt,
                                HiliteFuncPtr   func,
                                APPDrwCtx,
                                scSelection*&    select )
{
    REAL      dist;

    if ( !GetStream() )
        return;

    select = FlowsetGetSelection();
}

```

```

select->Restore( &mark, 0, false );

select->fPoint.fCol = this;

if ( Select( pt, &select->fPoint, &dist ) )
    select->LineHilite( func );
else
    raise( scERRlogical );
}

/* ===== */

void scColumn::InitialSelection( TypeSpec&      ts,
                                scSelection&& select )
{
    scMuPoint  mPt;
    TextMarker tm;
    REAL       dist;
    scContUnit* firstPara;
    Bool       iAdded = false;

    select = NULL;

    raise_if( GetPrev(), scERRlogical );

    if ( !GetStream() ) {
        firstPara = scContUnit::Allocate( ts, NULL, 0L );

        // initialize spec cache
        scCachedStyle::SetParaStyle( firstPara, ts );
        scCachedStyle::GetCachedStyle( ts );

        SetFlowsetStream( (scStream*)firstPara );

        Mark( scINVALID );
        LimitDamage( 0, scReformatTimeSlice );
        iAdded = true;
    }

    mPt.Set( 0, 0 );

    if ( !Select( mPt, &tm, &dist ) ) {
        if ( iAdded )
            FreeStream( );
        raise( scERRstructure );
    }

    select = FlowsetGetSelection();
    select->SetMark( tm );
    select->SetPoint( tm );
}

/* ===== */

void scColumn::LineInfo( scLineInfoList*  lineInfoList,
                        long&              nLines,
                        Bool&               moreText ) const
{
    scTextline* txl;

    nLines      = GetLinecount();
    moreText    = MoreText( );

    if ( lineInfoList && nLines ) {
        scLineInfo lineInfo;

        lineInfoList->RemoveAll();

        for ( txl = GetFirstline(); txl; txl = txl->GetNext() ) {
            txl->GetLineInfo( lineInfo );
            lineInfoList->AppendData( (ElementPtr)&lineInfo );
        }
    }
}

```

```

}

/* ===== */
/* paste a region into the indicated column, rebreak and return
 * damaged areas
 */

void scColumn::PasteRgn( const HRgnHandle  srcRgnH,
                        scRedisplist*    redisplist )
{
    HRgnHandle  dstRgnH;
    HRgn*       rgn;

    raise_if( srcRgnH == NULL, scERRstructure );

    dstRgnH = NewHRgn( RGNSliverSize( srcRgnH ) );

    if ( fRgnH ) {
        SectHRgn( fRgnH, srcRgnH, dstRgnH );
        DisposeHRgn( fRgnH );
    }
    else {
        CopyHRgn( dstRgnH, srcRgnH );
        SetShapeType( eRgnShape );
    }

    fRgnH = dstRgnH;
    scAutoUnlock h( fRgnH );
    rgn = (HRgn *)*h;

    fShapePieces = (ushort)rgn->fNumSlivers;

    Mark( scINVALID );
    LimitDamage( redisplist, scReformatTimeSlice );

    /* ===== */

void scColumn::CopyRgn( HRgnHandle& dstRgn )
{
    dstRgn = NewHRgn( RGNSliverSize( fRgnH ) );
    CopyHRgn( dstRgn, fRgnH );

    /* ===== */
    /* paste a polygon into the indicated column, rebreak and return
    * damaged areas
    */

void scColumn::ReplacePoly( scVertHandle  srcVertH,
                           scRedisplist*    redisplist )
{
    ushort          shapePieces;
    scVertex*       srcV;
    scVertex*       dstV;

    scAutoUnlock    h( srcVertH );
    srcV = (scVertex*)*h;

    shapePieces = POLYCountVerts( srcV );

    fVertH = MEMResizeHnd( fVertH, shapePieces * sizeof( scVertex ) );

    scAutoUnlock h1( fVertH );
    dstV = (scVertex*)*h1;
    SCmemmove( dstV, srcV, (size_t)(shapePieces * sizeof( scVertex )) );
    fShapePieces = shapePieces;

    /* check if poly type is set */
    SetShapeType( eVertShape );

```



```

    Mark( scINVALID );
    LimitDamage( redisplist, scReformatTimeSlice );
}

/* ===== */

#ifdef scColumnShape

/* add a polygon into the indicated column, rebreak and return
 * damaged areas
 */

void scColumn::PastePoly( scVertHandle srcVerth,
                          scRedisplist* redisplist )
{
    ushort          shapePieces;
    scVertex         *srcV,
                     *dstV;

    raise_if( GetShapeType() == eRgnShape, scERRstructure );

    scAutoUnlock h( srcVerth );
    srcV = (scVertex*)*h;

    shapePieces = POLYCountVerts( srcV );

    SetShapeType( eVertShape );

    fVerth = MEMResizeHnd( fVerth, shapePieces * sizeof(scVertex) );

    scAutoUnlock h2( fVerth );
    dstV = (scVertex*)*h2;

    if ( fShapePieces ) {
        dstV += ( fShapePieces - 1 );
        scAssert( dstV->fPointType == eFinalPoint );
        dstV->fPointType = eStopPoint;
        dstV++;
    }

    SCmemmove( dstV, srcV, (size_t)(shapePieces * sizeof( scVertex )) );
    fShapePieces = (ushort)(fShapePieces + shapePieces);

    Mark( scINVALID );
    LimitDamage( redisplist, scReformatTimeSlice );
}

/* ===== */

void scColumn::ClearShape( scRedisplist* redisplist )
{
    switch ( GetShapeType() ) {
        case eVertShape:
        case eRgnShape:
            SetShapeType( eNoShape );
            Mark( scINVALID );
            LimitDamage( redisplist, scReformatTimeSlice );
            break;
        case eVertFlex:
        case eHorzFlex:
        case eFlexShape:
        case eNoShape:
            break;
    }
}

/* ===== */

void scColumn::CopyPoly( scVertHandle* dstVerthP )
{
    scVertHandle     scrapPolyH;

```

```

    if ( fVertH && GetShape( ) == eVertShape ) {
        scrapPolyH = MEMAllocInd( fShapePieces * sizeof(scVertex) );

        scAutoUnlock    h1( scrapPolyH );
        scAutoUnlock    h2( fVertH );
        SCmemmove( (scVertex*)h1, (scVertex*)h2, (size_t)(fShapePieces * sizeof( scVertex ) ) );
    }
    else
        scrapPolyH = NULL;

    *dstVertHP = scrapPolyH;
}

/* ===== */

#endif

/* ===== */
/* this is primarily called when a column has changed, it forces a rebreak
 * of the paragraphs in the column, 'StrRerformat' should take care of damage
 * to subsequent paragraphs in subsequent columns, this also forces the
 * the rebreaking of any paragraphs that have no first line, thus if
 * a column is deleted it will force the correct rebreaking
 */

void scColumn::LimitDamage( scRedisplist* redisplist, long ticks )
{
    scContUnit* firstPara;
    scColumn*   nextcol;

    /* look thru the stream until we find an intersection of a paragraph
     * and a column, once we have an intersection we mark all the remaining
     * paragraphs to be rebroken, one problem is that if the column has been
     * made so small no lines are in it, then no paras are marked, the code
     * following the walk down the list takes care of that case
     */

    if ( !GetRecomposition() ) {
        Mark( scINVALID );
        return;
    }

    if ( !GetStream() )
        return;

    if ( Marked( scINVALID ) )
        firstPara = MarkParas( );
    else
        firstPara = GetStream();

    /* before we get into the stream make sure all paras that need to
     * be marked as REBREAK are marked as such
     */
    for ( nextcol = this; nextcol; nextcol = nextcol->GetNext() ) {
        if ( nextcol->Marked( scINVALID ) )
            nextcol->MarkParas();
    }

    scAssert( firstPara != 0 );

    STRReformat( this, firstPara, ticks, redisplist );
}

/* ===== */
/* renumber all the columns */

void scColumn::Renumber( )
{
    scColumn* col = (scColumn*)FirstInChain();
    long      count;

    /* renumber */
    for ( count = 0; col; col = col->GetNext() )

```

```

        col->SetCount( count );
    }

/* ===== */
/* creates a new unlinked empty column of specified width and depth */
scColumn::scColumn( APPColumn  appName,
                    MicroPoint  width,
                    MicroPoint  depth,
                    scStream*    p,
                    eCommonFlow flow ) :
    fShapePieces( 0 ),
    fRgnH( 0 ),
    fNextContext( 0 ),
    fName( appName ),
    fColumnCount( 0 ),
    fSize( width, depth ),
    fFlowDir( flow ),
    fStream( p ),
    fSelection( 0 ),
    fFirstline( 0 )
{
    SetShapeType( eNoShape );

    fInkExtents.Set( 0, 0, 0, 0 );

    /* add to context list */
    AddToCTXList();

    if ( appName == 0 )
        fName = (APPColumn)this;
}

/* ===== */
scColumn* scColumn::FindFlowset( const scStream* str )
{
    scColumn *col;

    col = fTheContextList;

    for ( ; col; col = col->GetContext() ) {
        if ( col->GetStream() == str )
            return col->GetFlowset();
    }
    return 0;
}

/* ===== */

void scColumn::DeleteFromCTXList( )
{
    scColumn *col;

    col = fTheContextList;

    if ( this == col )
        fTheContextList = GetContext();
    else {
        for ( ; col && col->GetContext() != this; col = col->GetContext() )
            ;
        if ( col )
            col->SetContext( GetContext() );
    }
}

/* ===== */

void scColumn::VerifyCTXList( void ) const
{
    register scColumn* col;

    for ( col = fTheContextList; col; col = col->GetContext() ) {

```

```

        if ( this == col )
            return;
    }

    raise( scERRidentification );
}

/* ===== */

void scColumn::FiniCTXList( void )
{
    scColumn*   col;
    scColumn*   nextCol;

    for ( col = fTheContextList; col; col = nextCol ) {
        SCDebugTrace( 1, scString( "FiniCTXList: 0x%08x\n" ), col );

        nextCol = col->GetContext();

        col->FreeStream( );

        // must do this since all layout are tracked
        col->scTBObj::Unlink();
        col->Free();
    }
}

/* ===== */

void scColumn::SetVertFlex( Bool      tf,
                           scRedisplList*  redisplList )

{
    if ( tf )
        SetShapeType( eVertFlex );
    else
        fLayBits.fLayType = (eColShapeType)( fLayBits.fLayType & ~eVertFlex );

    Mark( scINVALID );
    LimitDamage( redisplList, scReformatTimeSlice );
}

/* ===== */

void scColumn::SetHorzFlex( Bool      tf,
                           scRedisplList*  redisplList )
{
    if ( tf )
        SetShapeType( eHorzFlex );
    else
        fLayBits.fLayType = (eColShapeType)( fLayBits.fLayType & ~eHorzFlex );

    Mark( scINVALID );
    LimitDamage( redisplList, scReformatTimeSlice );
}

/* ===== */

void scColumn::SetShapeType( eColShapeType type )
{
    switch ( type ) {
        case eVertShape:
        case eRgnShape:
            if ( (eColShapeType)fLayBits.fLayType != type )
                FreeShape();
            fLayBits.fLayType = type;
            break;
        case eVertFlex:
        case eHorzFlex:
        case eFlexShape:
        case eNoShape:
            if ( (eColShapeType)fLayBits.fLayType & eIrregShape ) {
                /* we are trying to turn an irregularly shaped container

```

```

        * into a free container - we will have to free the shape
        */
        FreeShape();
    }
    if ( type == eNoShape )
        fLayBits.fLayType = type;
    else
        fLayBits.fLayType = (eColShapeType)(( fLayBits.fLayType & eFlexShape ) | type );
    break;
}
}

/* ===== */
/* free the lines with the column, this is tricky because we may want
 * to disentangle pointers at the same time
 */

void scColumn::FreeLines( Bool      reportDamage,
                          scXRect& lineDamage )
{
    scTextline* txl;
    scTextline* nextTxl;
    scContUnit* para;
    scXRect      extents;
    scContUnit* streamPresent = fStream;

    for ( txl = fFirstline; txl; txl = nextTxl ) {
#ifdef SCDEBUG > 1
        txl->scTBObj::scAssertValid();
#endif
        nextTxl = LNNNext( txl );
        if ( reportDamage ) {
            txl->QueryExtents( extents, 1 );
            if ( extents.Width() == 0 )
                extents.x2 = extents.x1 + 1;
            lineDamage.Union( extents );
        }
        if ( streamPresent != 0 ) {
            para = txl->GetPara( );
            if ( para && para->GetFirstline() == txl )
                para->SetFirstline( 0 );
        }
        delete txl;
    }

    SetFirstline( NULL );

    /* ===== */
    /* free the vertices of this column */

    void scColumn::FreeShape( )
    {
        switch ( GetShapeType() ) {
            case eVertShape:
                if ( fVertH != NULL )
                    MEMFreeHnd( fVertH );
                fShapePieces = 0;
                fVertH = NULL;
                break;
            case eRgnShape:
                if ( fRgnH != NULL )
                    DisposeHRgn( fRgnH );
                fShapePieces = 0;
                fRgnH = NULL;
                break;
            default:
                break;
        }
    }

    /* ===== */

```

```

/* free the column, no disemplement of pointers, save its own internal
 * structures
 */

```

```

void scColumn::Free()
{
    scXRect lineDamage;

    FreeLines( false, lineDamage );    // deletes lines
    SetShapeType( eNoShape );

    // free it up from the context list
    DeleteFromCTXList( );

    delete this;
}

```

```

/* ===== */
/* free the column that is part of the scrap */

```

```

void scColumn::FreeScrap( )
{
    scAssert( !GetNext() );

    FreeStream( );    /* deletes stream */

    DeleteFromCTXList( );
    Free();
}

```

```

/* ===== */
/* clear the stream from the set of linked columns,
 * that this column belongs to
 */

```

```

void scColumn::FlowsetClearStream( scRedisplList* redisplList )
{
    scColumn* firstCol = GetFlowset();
    scXRect lineDamage;

    // invalidate selection
    FlowsetInvalidateSelection();

    // free all the lines associated with the column(s)
    scColumn* col;
    for ( col = firstCol; col; col = col->GetNext() ) {
        if ( col->GetFirstline() )
            col->FreeLines( true, lineDamage );    /* deletes lines */
    }

    // delete the stream from all the column(s)
    firstCol->FreeStream( );
}

```

```

/* ===== */
/* cut the stream from the set of linked columns,
 * that this column belongs to
 */

```

```

void scColumn::FlowsetCutStream( scStream* stream,
                                scRedisplList* redisplList )
{
    scColumn* firstCol = GetFlowset();
    scXRect lineDamage;

    FlowsetInvalidateSelection();

    stream->STRDeformat();

    scColumn* col;
    for ( col = firstCol; col; col = col->GetNext() )
        col->FreeLines( true, lineDamage );    /* deletes lines */
}

```

```

    SetFlowsetStream( 0 );
}

/* ===== */

void scColumn::FlowsetPasteStream( scStream*      stream,
                                   scRedisplist*   redisplist )
{
    scColumn*   firstCol = GetFlowset();

    stream->STRMark( scREBREAK );

    if ( GetStream() )
        GetStream()->Append( stream );
    else
        SetFlowsetStream( stream );

    firstCol->Mark( scINVALID );
    firstCol->LimitDamage( redisplist, scReformatTimeSlice );
}

/* ===== */
/* free the column, not any text associated with it and unlink it from
 * its column chain
 */

void scColumn::Delete( scRedisplist* redisplist )
{
    scColumn* firstCol;
    scColumn* nextCol;

    firstCol      = (scColumn*)FirstInChain();
    nextCol       = GetNext();

    if ( this == firstCol ) {
        // trying to free a column in a chain without
        // unlinking it
        raise_if( nextCol && GetStream(), scERRstructure );

        if ( !nextCol && GetStream() ) {
            // we are the only column left so
            // we need to delete the text stream
            FreeStream();

            TypeSpec nullSpec;

            // clear the cache to help eliminate refs to specs
            scCachedStyle::StyleInvalidateCache( nullSpec );
        }
    }
    scTBObj::Unlink();
    DeleteFromCTXList();

    if ( firstCol != this ) {
        firstCol->Renumber();
        firstCol->Mark( scINVALID );
        firstCol->LimitDamage( redisplist, scReformatTimeSlice );
    }
    else if ( nextCol ) {
        firstCol->Renumber();
        firstCol->Mark( scINVALID );
        firstCol->LimitDamage( redisplist, scReformatTimeSlice );
    }

    Free();
}

/* ===== */
// because of reformatting nothing lands in here we will still return
// true

```

```

Bool scColumn::HasText( ) const
{
    scContUnit* p;

    for ( p = GetStream(); p; p = p->GetNext( ) ) {
        if ( p->GetContentSize() > 0 )
            return true;
    }
    return false;
}

/* ===== */
/* does the text flow out the bottom of this container */
/* ===== */

Bool scColumn::MoreText( ) const
{
    scTextline* txl;
    scContUnit* para;
    scColumn* neighborCol;

    txl = GetLastline();

    if ( txl ) {
        para = txl->GetPara();
        if ( para->GetNext() )
            return true;
        else if ( para->GetContentSize() > txl->GetEndOffset() )
            return true;
    }
    else if ( GetStream() ) {
        neighborCol = NextWithLines(); // text in subsequent columns
        if ( neighborCol )
            return true;

        neighborCol = PrevWithLines();

        if ( neighborCol ) {
            txl = neighborCol->GetLastline();

            // this gets a little tricky, we are assuming that
            // the text cannot be reformatted into this or
            // some other column and therefore it hangs off the
            // end
            para = txl->GetPara();
            if ( para->GetNext() )
                return true; // another paragraph beyond last formatted line
            else if ( para->GetContentSize() > txl->GetEndOffset() )
                return true; // more characters beyond last formatted line
            return false; // no more text
        }
        return true; // no text formatted and we have a stream
    }
    return false;
}

/* ===== */
// determines line num in column of selection, assumes a sliver cursor
// ===== */

short COLLineNum( scSelection* select )
{
    scColumn* col;
    scTextline* txl;
    scTextline* countTxl;
    short lineCount;

    if ( select ) {
        col = select->fMark.fCol;
        txl = select->fMark.fTxl;
        if ( col && txl ) {
            countTxl = col->GetFirstline();

```



```

        for ( lineCount =
            countTx1 != NULL;
              lineCount++, countTx1 = LNNext( countTx1 ) ) {
            if ( countTx1 == tx1 )
                return lineCount;
        }
    }
    return -1;
}

```

```

/* ===== */
/* determine the size of the damagerect for ImmediateRedisp depending
 * on the lines set
 */

```

```

void scColumn::LineExtents( scImmediateRedisp& immediateRedisp )
{
    scTextline* tx1;
    short      count;
    scXRect     colRect;
    scXRect     rect;

    colRect.Invalidate();

    tx1 = fFirstline;
    for ( count = 1; tx1 && count < immediateRedisp.fStartLine; count++ )
        tx1 = tx1->GetNext();

    if ( tx1 ) {
        do {
            tx1->QueryExtents( rect );
            colRect.Union( rect );
            tx1 = tx1->GetNext();
            count++;
        } while ( tx1 && count <= immediateRedisp.fStopLine );
    }

    if ( colRect.Valid() ) {
        if ( fFlowDir.IsHorizontal() ) {
            colRect.x1 = MIN( colRect.x1, 0 );
            colRect.x2 = MAX( colRect.x2, Width() );
        }
        else {
            colRect.y1 = MIN( colRect.y1, 0 );
            colRect.y2 = MAX( colRect.y2, Depth() );
        }
    }

    immediateRedisp.fImmediateRect = colRect;
}

```

```

/* ===== */
/* draw the line of text in the selection */

```

```

void scColumn::UpdateLine( scImmediateRedisp& immediateRedisp,
                          APPDrwCtx          mat )
{
    scTextline* paintTx1;
    short      count;
    scMuPoint   tx( 0, 0 );

    paintTx1 = fFirstline;
    for ( count = 1; paintTx1 != NULL && count < immediateRedisp.fStartLine; count++ )
        paintTx1 = paintTx1->GetNext();

    if ( paintTx1 != NULL ) {
        do {
            paintTx1->Draw( mat, GetFlowdir(), tx );
            paintTx1 = paintTx1->GetNext();
            count++;
        } while ( paintTx1 != NULL && count <= immediateRedisp.fStopLine );
    }
}

```

```

}

/* ===== */
/* draw the portions of the column that intersect the 'damagedRectangle' */

void scColumn::Draw( const scXRect&      dRect,
                    APPDrwCtx          dc,
                    const scMuPoint*    translation )
{
    scTextline* txl;
    scXRect      exRect;
    scMuPoint     tx( 0, 0 );

    if ( translation )
        tx += *translation;

    for ( txl = GetFirstline(); txl != NULL; txl = txl->GetNext() ) {
        txl->QueryExtents( exRect, 1 );
        if ( exRect.Intersect( dRect ) ) {
            txl->Draw( dc, fFlowDir, tx );
            txl->Unmark( scREPAINT );
        }
    }
}

/* ===== */
/* read from a text file */

void scColumn::ReadTextFile( TypeSpec      spec,
                             APPCtxPtr     ctxPtr,
                             IOFuncPtr     readFunc,
                             scRedisplList* redisplList )
{
    scColumn*      startCol;

    scCachedStyle::SetFlowdir( GetFlowdir() );
    scCachedStyle::GetCachedStyle( spec );

    startCol = (scColumn*)FirstInChain();

    if ( GetStream() )
        GetStream()->RemoveEmptyTrailingParas( GetFlowset() );

    if ( startCol->GetStream() == NULL )
        SetFlowsetStream( scStream::ReadTextFile( spec, ctxPtr, readFunc, 0 ) );
    else
        startCol->GetStream()->Append( scStream::ReadTextFile( spec, ctxPtr, readFunc, 0 ) );

    startCol->Mark( scINVALID );
    startCol->LimitDamage( redisplList, scReformatTimeSlice );    /* reBreak */
}

/* ===== */
/* paste APPText into a text container */

void scColumn::PasteAPPText( stTextImportExport& appText,
                             scRedisplList* redisplList )
{
    scColumn*      firstCol;
    TypeSpec       nullSpec;

    if ( GetStream() )
        GetStream()->RemoveEmptyTrailingParas( GetFlowset() );

    firstCol = GetFlowset();

    if ( !fStream )
        firstCol->SetFlowsetStream( scStream::ReadAPPText( appText ) );
    else
        fStream->Append( scStream::ReadAPPText( appText ) );

    firstCol->Mark( scINVALID );
    firstCol->LimitDamage( redisplList, scReformatTimeSlice );    /* reBreak */
}

```

```

}

/* ===== */
/* upon completion of reading data in from disk we search down the column list
 * finding the first columns in a chain and retabulate, rebreak and repaint
 */

void scColumn::Update( scRedisList *redisList )
{
    scColumn*   flowset;
    scColumn*   col;

    for ( col = GetBaseContextList(); col; col = col->GetContext() ) {
        if ( col->Marked( scINVALID ) && col->GetRecomposition() ) {
            flowset = (scColumn*)col->FirstInChain();

            scCachedStyle::SetFlowdir( flowset->GetFlowdir() );

            flowset->LimitDamage( redisList, scReformatTimeSlice );

            scColumn* p = flowset;
            for ( ; p; p = p->GetNext() )
                p->Unmark( scINVALID );
        }
    }

    /* ===== */
    /* this is still a little dirty - needs to be cleaned up a bit */
    /* reformat all columns containing ts */

void scColumn::ChangedTS( TypeSpec      theTS,
                        eSpecTask      task,
                        scRedisList* redisList )

{
    scColumn*   aColH;
    scContUnit* p;
    scTextline* txl;

    scCachedStyle::StyleInvalidateCache( theTS );

    for ( aColH = GetBaseContextList(); aColH; aColH = aColH->GetContext() ) {
        if ( aColH->GetCount() == 0 ) {

            scCachedStyle::SetFlowdir( aColH->GetFlowdir() );

            p = aColH->GetStream();
            for ( ; p; p = p->GetNext() ) {
                if ( p->ContainTS( theTS ) ) {
                    if ( !aColH->GetRecomposition() ) {
                        if ( ( txl = p->GetFirstline() ) != NULL ) {
                            scColumn * colH;

                            if ( ( colH = txl->GetColumn() ) != NULL )
                                colH->Mark( scINVALID );
                        }
                        else
                            aColH->Mark( scINVALID );

                        if ( task & eSCRetabulate )
                            p->Mark( scRETABULATE );
                        if ( task & eSCRebreak )
                            p->Mark( scREBREAK );
                        if ( task & eSCRepaint )
                            p->ForceRepaint( 0L, LONG_MAX );
                    }
                    else {
                        if ( task & eSCRetabulate )
                            p->Retabulate( theTS );
                        if ( task & eSCRebreak )
                            p->Mark( scREBREAK );
                        if ( task & eSCRepaint )
                            p->ForceRepaint( 0L, LONG_MAX );
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    if ( aColH->GetRecomposition() && aColH->GetStream() )
        STRReformat( aColH, aColH->GetStream(), scReformatTimeSlice, redisplList );
    else {
        scSelection* select = aColH->FlowsetGetSelection();
        select->UpdateSelection( );
    }
}

}

/* ===== */
/* search a column building a list of typespecs that are contained
 * in the column
 */

void scColumn::GetTSList( scTypeSpecList& tsList ) const
{
    scTextline* txl;

    for ( txl = GetFirstline(); txl; txl = txl->GetNext() )
        txl->GetTSList( tsList );
}

/* ===== */
/* determine the prev column with a line in it */
scColumn* scColumn::PrevWithLines() const
{
    scColumn* col;

    for ( col = GetPrev(); col ; col = col->GetPrev() ) {
        if ( col->GetFirstline( ) )
            return col;
    }
    return 0;
}

/* ===== */
/* determine the next column with a line in it */
scColumn* scColumn::NextWithLines( ) const
{
    scColumn* col;

    for ( col = GetNext(); col; col = col->GetNext( ) ) {
        if ( col->GetFirstline( ) )
            return col;
    }
    return 0;
}

/* ===== */
/* return the last line in this column */

scTextline* scColumn::GetLastline( ) const
{
    scTextline* txl;
    scTextline* validLine = 0;

    for ( txl = fFirstline; txl; txl = txl->GetNext() ) {
        if ( !txl->Marked( scINVALID ) ) {
            validLine = txl;
            // validLine->AssertValid();
        }
    }
    return validLine;
}

```

```

/* ===== */
/* mark all the paras contained within this container to be rebroken */

scContUnit* scColumn::MarkParas( )
{
    scContUnit* firstPara;
    scContUnit* lastPara;
    scContUnit* para;
    scColumn*   contentCol;

    firstPara = FirstPara();

    if ( firstPara ) {
        // in this case the container has some lines
        lastPara = LastPara();
        for ( para = firstPara; para; para = para->GetNext() ) {
            para->Mark( scREBREAK );
            if ( para == lastPara )
                break;
        }
    }
    else {
        /* in this case the container has no lines,
        * we must try to find a neighbor that has
        * some lines, first we look backwards and then
        * we look forwards, we mark what we find and
        * see if they will reformat into the container
        */
        if ( !GetPrev() )
            firstPara = GetStream();
        else {
            contentCol = PrevWithLines();
            if ( contentCol )
                firstPara = contentCol->LastPara();
            else {
                contentCol = NextWithLines( );
                if ( contentCol )
                    firstPara = contentCol->FirstPara();
            }
            if ( !firstPara ) {
                /* this would be executed if no containers had lines
                * attached to them
                */
                firstPara = GetStream();
            }
        }
        if ( firstPara )
            firstPara->Mark( scREBREAK );
    }
    return firstPara;
}

/* ===== */
/* return the paragraph of the last line of text in this column */

scContUnit* scColumn::LastPara( ) const
{
    scTextline* txl = GetLastline();

    for ( ; txl && txl->Marked( scINVALID ); txl = txl->GetPrev() )
        ;

    return txl ? txl->GetPara() : NULL;
}

/* ===== */
// return the first valid paragraph of this column

scContUnit* scColumn::FirstPara( ) const
{
    // if no previous column the first guy in the stream is
    // the first paragraph
    if ( !GetPrev() )

```

```
return fStream ? fStream->First() : 0;
```

```
scColumn*   prev      = GetPrev();
scTextline* txl        = 0;
scContUnit* p          = 0;
```

```
// get last valid line in prev para, presumably
// the container has been reformatted
```

```
do {
    txl = prev->GetLastline();
    if ( !txl )
        prev = prev->GetPrev();
} while ( prev && !txl );
```

```
// get the paragraph of the last line, check to
// see if the end of the line represents the end
// of the paragraph, if it does go to the next para
```

```
if ( txl ) {
    p = txl->GetPara();

    if ( txl->GetEndOffset() == p->GetContentSize() )
        p = p->GetNext();
}
else
    p = fStream ? fStream->First() : 0;
```

```
return p;
```

```
/* ===== */
/* Delete excess lines in the column */
```

```
void scColumn::DeleteExcessLines( scContUnit*   para,
                                   scTextline*   lastTxl,
                                   Bool           testGetStrip,
                                   scCOLRefData& colReformatData )
```

```
scTextline* txl;
scTextline* nextTxl;
Bool        deleteLines = false;
```

```
if ( lastTxl ) {
    if ( ( txl = LNNext( lastTxl ) ) != NULL )
        deleteLines = true;
}
else if ( ( txl = GetFirstline() ) != NULL ) {
```

```
    if ( para == NULL || para->GetCount() <= txl->GetPara()->GetCount() ) {
        if ( ! testGetStrip )
            deleteLines = true;
        else {
            scLINERefData lineData;
```

```
            scCachedStyle::SetFlowdir( GetFlowdir() );
            TypeSpec ts = txl->SpecAtStart();
            scCachedStyle::GetCachedStyle( ts );
```

```
            lineData.fOrg          = txl->GetOrigin();
            lineData.fMeasure      = txl->GetMeasure();
            lineData.fLogicalExtents = scCachedStyle::GetCurrentCache().GetLogicalExtents( );
            lineData.fInitialLead.Set( scCachedStyle::GetCurrentCache().GetComputedLead(), scCachedStyle::GetCurrentCache().GetFlowdir() );
```

```
            if ( !GetStrip( lineData, eStartColBreak, colReformatData ) )
                /* the first line will not fit, delete them */
                deleteLines = true;
        }
    }
```

```
    }
}
```

```
if ( deleteLines ) {
    Mark( scREPAINT );      /* if we delete we need to repaint */
}
```

```

        for ( ; tx1; tx1 = nextTx1 ) {
            nextTx1 = tx1->GetNext();
            tx1->Delete( colReformatData.fLineDamage );
        }
    }

/* ===== */

void scColumn::SetFlowsetStream( scStream* cu )
{
    scColumn* col;

    for ( col = (scColumn*)FirstInChain(); col; col = col->GetNext() )
        col->SetStream( cu );
}

/* ===== */
/* free the stream from the column chain */

void scColumn::FreeStream()
{
    if ( fStream ) {
        fStream->STIRFree();
        SetFlowsetStream( 0 );
    }
}

/* ===== */
/* force the rebreaking of this column */

void scColumn::Rebreak( scRedisplist* redisplist )
{
    // save the recomposition state
    Bool    saveRecomposeFlag = GetRecomposition();

    SetRecomposition( true );

    Rebreak2( redisplist );

    // restore the saved value
    SetRecomposition( saveRecomposeFlag );

    Unmark( scINVALID );
}

/* ===== */
/* rebreak of this column */

void scColumn::Rebreak2( scRedisplist* redisplist )
{
    Mark( scINVALID );

    if ( DamOpen() )
        LimitDamage( redisplist, scReformatTimeSlice );
}

/* ===== */
/* give the column a new width & depth, rebreak and return damaged areas */
/* the column measure and/or depth has changed respond accordingly
 * OBVIOUS OPTIMIZATIONS
 * if depth increases just add stuff
 */

void scColumn::Resize( MicroPoint    width,
                      MicroPoint    depth,
                      scRedisplist*  redisplist )
{
    switch ( GetShapeType() ) {
        case eRgnShape:
        case eVertShape:
            SetSize( width, depth );
            return;
    }
}

```

```

        case eNoShape:
            SetSize( width, depth );
            break;
        case eVertFlex:
            SetWidth( width );
            break;
        case eHorzFlex:
            SetDepth( depth );
            break;
        case eFlexShape:
            SetSize( width, depth );
            break;
    }
    Mark( scINVALID );
    LimitDamage( redisplist, scReformatTimeSlice );
}

/* ===== */

void scColumn::Resize( const scSize&    newSize,
                      scRedisList*    redisplist )
{
    switch ( GetShapeType() ) {
        case eRgnShape:
        case eVertShape:
            SetSize( newSize );
            return;
        case eNoShape:
            SetSize( newSize );
            break;
        case eVertFlex:
            SetWidth( newSize.Depth() );
            break;
        case eHorzFlex:
            SetDepth( newSize.Width() );
            break;
        case eFlexShape:
            SetSize( newSize );
            break;
    }
    Mark( scINVALID );
    LimitDamage( redisplist, scReformatTimeSlice );
}

/* ===== */
/* ENUMERATE THE COLUMN AND ITS STRUCTURES */

void scColumn::Enumerate( long& objEnumerate )
{
    scTBObj::Enumerate( objEnumerate );

    // if the column has no previous members, that is it is
    // the first column of a set of linked columns, enumerate
    // the paragraphs and their text
    //
    if ( !Prev() && fStream )
        fStream->DeepEnumerate( objEnumerate );
}

/* ===== */

/* return the size of this column for storage purposes, the text stream
 * is always stored with the first column, subsequent columns store
 * just the container itself. (this may present problems for paging of text
 * in multipage documents)
 */

void scColumn::ExternalSize( long& exSize )
{
    scContUnit* para;

```



```

exSize = sizeof(scColumn)

if ( !GetPrev() ) {
    for ( para = GetStream(); para; para = para->GetNext( ) )
        exSize += para->ExternalSize();
}
switch ( GetShapeType() ) {

    case eVertShape:
#ifdef ColumnPolygon
        exSize += POLYExternalSize( fVertH, fShapePieces );
#endif /* ColumnPolygon */
        break;

    case eRgnShape:
        exSize += RGNExternalSize( fRgnH, fShapePieces );
        break;
}
exSize += sizeof( scTBObj );      /* NULL OBJECT */
}

/* ===== */

void scColumn::ZeroEnumeration( )
{
    ZeroEnum();

    if ( !GetPrev() )
        GetStream()->STRZeroEnumeration();

/* ===== */
/* determine extents of the column in its local coordinates */

void scColumn::ComputeInkExtents( )
{
    scXRect    lineExtents;
    scTextline* txl;

    /* clear rect */
    fInkExtents.Set( 0, 0, 0, 0 );
    /* add each line to the current extents */
    for ( txl = fFirstline; txl; txl = LNNext( txl ) ) {
        txl->QueryExtents( lineExtents, 1 );
        if ( lineExtents.Width() <= 0 )
            lineExtents.x2 = lineExtents.x1 + 1;
        fInkExtents.Union( lineExtents );
    }
}

/* ===== */
/* determine extents of the column in its local coordinates */

static void COLQueryMarginsVertical( const scColumn*    col,
                                     scXRect&          margins,
                                     int                shapeType )
{
    scTextline *txl;
    scMuPoint  translate;
    TypeSpec   spec;
    scXRect    xrect2;

    switch ( shapeType ) {
        case eHorzFlex:
            txl = col->GetFirstline();

            if ( txl ) {
                margins.Set( txl->GetOrigin().x,
                           txl->GetOrigin().y,
                           txl->GetOrigin().x + CSfirstLinePosition( col->GetAPPName(), txl->SpecAtStart( ) ),
                           col->Depth() );
            }
    }
}

```

```

        for ( txl = col->GetFirstline(); txl; txl = LNNext( txl ) ) {
            xrect2.Set( txl->GetOrigin().x,
                        txl->GetOrigin().y,
                        txl->GetOrigin().x + CSfirstLinePosition( col->GetAPPName(), tx
l->SpecAtStart( ) ),
                        col->Depth() );
            margins.Union( xrect2 );
        }

        txl = col->GetLastline( );
        txl->MaxLead( spec );
        xrect2.Set( txl->GetOrigin().x - CSlastLinePosition( col->GetAPPName(), spec ),
                    txl->GetOrigin().y,
                    txl->GetOrigin().x,
                    col->Depth() );

        margins.Union( xrect2 );
    }
    break;

case eFlexShape:
    txl = col->GetFirstline();
    if ( txl ) {
        margins.Set( txl->GetOrigin().x,
                    txl->GetOrigin().y,
                    txl->GetOrigin().x + CSfirstLinePosition( col->GetAPPName(), txl->S
pecAtStart( ) ),
                    txl->GetMeasure() );

        for ( txl = col->GetFirstline(); txl; txl = LNNext( txl ) ) {
            xrect2.Set( txl->GetOrigin().x,
                        txl->GetOrigin().y,
                        txl->GetOrigin().x + CSfirstLinePosition( col->GetAPPName(), tx
->SpecAtStart( ) ),
                        txl->GetMeasure() );
            margins.Union( xrect2 );
        }

        txl = col->GetLastline( );
        txl->MaxLead( spec );
        xrect2.Set( txl->GetOrigin().x - CSlastLinePosition( col->GetAPPName(), spec ),
                    txl->GetOrigin().y,
                    txl->GetOrigin().x,
                    txl->GetMeasure() );

        margins.Union( xrect2 );
    }
    break;

case eVertFlex:
    margins.Set( 0, 0, col->Width(), 0 );

    for ( txl = col->GetFirstline(); txl; txl = txl->GetNext() )
        margins.y2 = MAX( txl->GetOrigin().y + txl->GetLength(), margins.y2 );
    break;
}
}

void scColumn::QueryMargins( scXRect& margins ) const
{
    scTextline *txl;
    scTextline *nextTxl;
    TypeSpec spec;

    if ( GetFlowdir().IsVertical() ) {
        switch ( GetShapeType() ) {
            case eHorzFlex:
            case eVertFlex:
            case eFlexShape:
                COLQueryMarginsVertical( this, margins, GetShapeType() );
                return;
            default:

```

```

        break;
    }
}

switch ( GetShapeType() ) {
    case eFlexShape:
    case eVertFlex:

        if ( GetShapeType() == eFlexShape )
            margins.Set( 0, 0, 0, 0 );
        else
            margins.Set( 0, 0, Width(), 0 );

        /* add each line to the current extents */
        for ( txl = GetFirstline(); txl; txl = nextTxl ) {

            if ( GetShapeType() == eFlexShape )
                margins.x2 = MAX( txl->GetOrigin().y + txl->GetLength(), margins.x2 );

            nextTxl = LNNext( txl );
            if ( !nextTxl ) { /* last line */

                margins.y2 = MAX( txl->GetOrigin().y, margins.y2 );

                /* this makes vertical flex columns the size
                 * of the text baseline plus whatever amount
                 * of text the application wants to add to the bottom
                 */
                MicroPoint maxlead = txl->MaxLead( spec );
                if (spec.ptr()) {
                    margins.y2 += CSlastLinePosition( GetAPPName(), spec );
                }

                margins.y2 += txl->GetVJOffset();
            }
        }
        break;

    case eHorzFlex:

        margins.Set( 0, 0, 0, Depth() );

        for ( txl = GetFirstline(); txl; txl = txl->GetNext() )
            margins.x2 = MAX( txl->GetOrigin().x + txl->GetLength(), margins.x2 );
        break;

    case eVertShape:
    case eRgnShape:
    case eNoShape:
        margins.Set( 0, 0, Width(), Depth() );
        break;
}

}

/* ===== */
/* determine maximum possible depth of the column in its local coordinates */

void scColumn::QuerySize( scSize& size ) const
{
    switch ( GetShapeType() ) {

#ifdef ColumnPolygon
        case eVertShape:
            size.SetDept( POLYMaxDepth( fVertH ) );
            break;
#endif /* ColumnPolygon */

        case eRgnShape:
        {
            scXRect xrect;

            RGNGetExtents( fRgnH, xrect );
            // this is open to some discussion

```

```

        // over width is correct
        size.SetWidth( xrect.x2 );
        size.SetDepth( xrect.y2 );
    }
    break;

    case eVertFlex:
    case eFlexShape:
        size.SetWidth( Width() );
        size.SetDepth( LONG_MAX );
        break;

    default:
    case eHorzFlex:
    case eNoShape:
        size = GetSize();
        break;
    }
}

/* ===== */
/* Determine maximum depth of text from top (or from right in vertical) */

void scColumn::QueryTextDepth( MicroPoint& depth ) const
{
    switch ( GetShapeType() ) {
        case eVertShape:
            depth = POLYMaxDepth( fVertH );
            break;

        case eRgnShape:
            depth = RGNMaxDepth( fRgnH );
            break;

        case eVertFlex:
            if ( GetFlowdir().IsVertical() ) {
                depth = TextDepth();
                break;
            }

        case eFlexShape:
            depth = LONG_MAX;
            break;

        case eHorzFlex:
            if ( GetFlowdir().IsVertical() ) {
                depth = LONG_MAX;
                break;
            }

        default:
        case eNoShape:
            depth = TextDepth();
            break;
    }
}

/* ===== */

MicroPoint scColumn::TextDepth( ) const
{
    return GetFlowdir().IsHorizontal() ? Depth() : Width();
}

/* ===== */

static Bool COLLinkSetContains( scColumn * col1H,
                                scColumn * col2H )
{
    scColumn * prevColH;

    /* backup */
    for ( ; col1H && (prevColH = col1H->GetPrev()) != NULL;
          col1H = prevColH )
        ;
}

```

```

/* renumber */
for ( ; col1H; col1H = col1H->GetNext() ) {
    if ( col1H == col2H )
        return true;
}
return false;
}

/* ===== */

void scColumn::Link( scColumn*    col2,
                    Bool          reformat,
                    scRedisList*  redisList )
{
    scSelection*    select2 = 0;

    // make sure the existing links make sense
    raise_if ( col2->GetPrev(), scERRstructure );

    raise_if( COLLinkSetContains( this, col2 ), scERRstructure );

    /* mark the paras in each to be rebroken */
    MarkParas( );
    col2->MarkParas( );

    col2->FlowsetSetFlowdir( GetFlowdir() );

    if ( FlowsetGetSelection() && !col2->GetSelection() )
        ; // we are cool
    else if ( !FlowsetGetSelection() && col2->GetSelection() ) {
        // transfer selection
        select2 = col2->FlowsetGetSelection();
        col2->FlowsetRemoveSelection();
        FlowsetSetSelection( select2 );
    }
    else {
        select2 = col2->FlowsetGetSelection();
        col2->FlowsetRemoveSelection();
        delete select2, select2 = 0;
    }

    // do the actual link
    scTBObj::Link( col2 );

    /* patch the stream(s)
     * if either column has a stream we can deal with it easily.
     * if both have it, append stream2 to stream1
     */
    if ( GetStream() && !col2->GetStream() ) {
        /* col1 has a stream */
        SetFlowsetStream( GetStream() );
    }
    else if ( col2->GetStream() && !GetStream() ) {
        /* col2 has a stream */
        SetFlowsetStream( col2->GetStream() );
    }
    else if ( GetStream() && col2->GetStream() ) {
        // both contain streams
        GetStream()->Append( col2->GetStream() );
        SetFlowsetStream( GetStream() );
    }
    else
        /* no column has a stream */;

    // renumber the streams
    Renumber();

    // patch selection

    if ( reformat ) {
        Mark( scINVALID );
        LimitDamage( redisList, scReformatTimeSlice );
    }
}

```

```

    }
}

/* ===== */

void scColumn::Unlink( scRedisList* redisList )
{
    scColumn*    firstCol;
    scXRect      lineDamage;

    // mark the paras in the container beings unlinked to be rebroken,
    // since they are losing their home, they definately need to
    // be rebroken
    //
    firstCol = GetPrev();
    if ( firstCol == NULL )
        firstCol = GetNext();

    if ( firstCol ) {
        MarkParas();
        FreeLines( true, lineDamage ); /* deletes lines */

        if ( redisList )
            redisList->AddColumn( this, lineDamage );

        scTBObj::Unlink( );
        SetFlowsetStream( 0 );

        firstCol->Renumber( );
        firstCol->Mark( scINVALID );
        firstCol->LimitDamage( redisList, scReformatTimeSlice );
    }
}

/* ===== */

void scColumn::BreakChain( scColumn* col2 )
{
    raise_if( GetNext() != col2, scERRstructure );

    if ( GetStream() )
        GetStream()->STRDeformat(); /* remove any layout information

        // break the link
        SetNext( 0 );
        col2->SetPrev( 0 );

        col2->SetFlowsetStream( 0 ); /* set the stream in col 2 to nothing

/* ===== */

void scColumn::InvertExtents( HiliteFuncPtr func,
                             APPDrwCtx      mat )
{
    scTextline* txl;

    for ( txl = GetFirstline( ); txl; txl = txl->GetNext() )
        txl->InvertExtents( func, mat );
}

/* ===== */
/* set the flow direction of the container */

void scColumn::FlowsetSetFlowdir( const scFlowDir& flowDir )
{
    scColumn*    col = GetFlowset();

    for ( ; col != 0; col = col->GetNext() ) {
        if ( col->GetFlowdir() != flowDir ) {
            col->SetFlowdir( flowDir );
            col->Mark( scINVALID );
            scStream* str = col->GetStream();

```

```

        if ( str )
            col->GetStream()->STRMark( scRETABULATE | scREBREAK );
    }
    scCachedStyle::SetFlowdir( flowDir );
    GetFlowset()->LimitDamage( 0, scReformatTimeSlice );
}

/* ===== */

void scFlowDir::SetFlow( eCommonFlow cf )
{
    if ( cf == eNoFlow ) {
        linedir_ = eInvalidFlow;
        glyphdir_ = eInvalidFlow;
    }
    else if ( cf == eRomanFlow ) {
        linedir_ = eTopToBottom;
        glyphdir_ = eLeftToRight;
    }
    else if ( cf == eVertJapanFlow ) {
        linedir_ = eRightToLeft;
        glyphdir_ = eTopToBottom;
    }
    else if ( cf == eBidiFlow ) {
        linedir_ = eTopToBottom;
        glyphdir_ = eRightToLeft;
    }
}

/* ===== */

eCommonFlow scFlowDir::GetFlow() const
{
    if ( linedir_ == eTopToBottom && glyphdir_ == eLeftToRight )
        return eRomanFlow;
    else if ( linedir_ == eRightToLeft && glyphdir_ == eTopToBottom )
        return eVertJapanFlow;
    else if ( linedir_ == eTopToBottom && glyphdir_ == eRightToLeft )
        return eBidiFlow;
    return eNoFlow;
}

/* ===== */

#ifdef SCDEBUG > 1
void scColumn::DbgPrintInfo( int debugLevel ) const
{
    SCDebugTrace( debugLevel, scString( "\nSCC COLUMN 0x%08x - firstline 0x%08x\n" ), this, fFirstline );

    scTextline* txl;

    for ( txl = fFirstline; txl; txl = txl->GetNext() )
        txl->DbgPrintInfo( debugLevel );
}

/* ===== */

#endif

```

File: SCMACINT.H

\$Header: /Projects/Toolbox/ct/SCMACINT.H 2 5/30/97 8:45a Wmanis \$

Contains: Defines for MacIntosh/MPW compile

Written by:

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

```

=====*/

#ifndef _H_SCMACINT
#define _H_SCMACINT

/*
 * Header configuration. To determine if we are using universal headers load types.h and then
 * look to see if it has included the universal headers <ConditionalMacros.h> file.
 */
#include <Types.h>

#ifdef __CONDITIONALMACROS__
#define USEUNIVERSALHEADERS 1
#else
#define USEUNIVERSALHEADERS 0
#endif

#if USEUNIVERSALHEADERS && !defined(USESROUTINEDESCRIPTORS)
#define USESROUTINEDESCRIPTORS 1
#endif

/* SYSTEM INCLUDES */
#include "StdDef.h"

// #include <OSUtils.h>
// #include <Events.h>
// #include <limits.h>
#include <string.h>
// #include <math.h>

// memory model stuff - for intel only
#define scNEAR
#define scFAR
#define SCHuge

// volatile is not supported by MPW
#define volatile

#define SCTickCount() (TickCount())
#define SCSysBeep(duration) (SysBeep((int)(duration)))

#endif /* _H_SCMACINT.H */

```



```

uchar*      BufSet_REAL( u_char* rbuf[12],
                        REAL      r,
                        eByteOrder desiredByteOrder );

const uchar* BufGet_REAL( const uchar* rbuf[12],
                        REAL&          pr,
                        eByteOrder      byteOrder );

// the follow are not good for
// writing out alot of data, but for a long
// here are there they are goo
void ReadLong( long&,
              APPCtxPtr,
              IOFuncPtr,
              eByteOrder );

// a quick way of writing out a long
void WriteLong( long,
              APPCtxPtr,
              IOFuncPtr,
              eByteOrder );

void ReadBytes( uchar*,
              APPCtxPtr,
              IOFuncPtr,
              long );

void WriteBytes( const uchar*,
              APPCtxPtr,
              IOFuncPtr,
              long );
#endif

```

File: pfileio.h

\$Header: /Projects/Toolbox/ct/SCFILEIO.H 2 5/30/97 8:45a Wmanis \$

Contains: Independent byte order calls.

Written by: Coletti

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

```

*****/

#ifndef _H_PFILEIO
#define _H_PFILEIO

#include "sctypes.h"

typedef enum eByteOrders {
    kNoOrder      = 0,
    kIntelOrder   = 1,
    kMotorolaOrder = 2
} eByteOrder;

typedef uchar  REALBUF[12];
typedef uchar  ByteOrderStr[8];

#define kShortBufSize  2
#define kLongBufSize   4

uchar*      BufSet_byteorder( uchar[] );
const uchar* BufGet_byteorder( const uchar[], short* );

uchar*      BufSet_char( uchar*      dstbuf,
                        const uchar*  srcbuf,
                        size_t        bytes,
                        eByteOrder    desiredByteOrder );

const uchar* BufGet_char( const uchar*  srcbuf,
                        uchar*          dstbuf,
                        size_t          bytes,
                        eByteOrder      byteOrder );

uchar*      BufSet_short( uchar      sbuf[2],
                        ushort        s,
                        eByteOrder    desiredByteOrder );

const uchar* BufGet_short( const uchar  sbuf[2],
                        ushort&          ps,
                        eByteOrder      byteOrder );

uchar*      BufSet_long( uchar      pbuf[4],
                        ulong         l,
                        eByteOrder    desiredByteOrder );

const uchar* BufGet_long( const uchar  lbuf[4],
                        ulong&          pl,
                        eByteOrder      byteOrder );

```

File: SCGLOBDA.C

\$Header: /Projects/Toolbox/ct/SCGLOBDA.CPP 2 5/30/97 8:45a Wmanis \$

Contains: Global data, which should be gone soon!

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

*****/
#include "scexcept.h"

#include <string.h>
#include "scmem.h"

#include "scglobda.h"
#include "scparagr.h"
#include "sccolumn.h"
#include "sctextli.h"

scDEFINE_RTTI(scTBObj, scObject);
scDEFINE_RTTI(scColumn, scTBObj);
scDEFINE_RTTI(scTextline, scTBObj);
scDEFINE_RTTI(scContUnit, scTBObj);

scDEFINE_ABSTRACT_RTTI(scAbstractArray, scObject);
scDEFINE_RTTI(scHandleArray, scAbstractArray);
scDEFINE_RTTI(scMemArray, scAbstractArray);
scDEFINE_RTTI(scCharArray, scHandleArray);

char *SCS_Copyright = "Copyright (c) 1988-1994 Stonehand Inc. All rights reserved.";

BreakStruct gbrS;
GlobalColumnStruct ggcS;
scStreamChangeInfo gStreamChangeInfo;

Bool gHiliteSpaces; // hilite trailing spaces at the end of a line

long scLogUnitsPerPixel = 20;

```

        scMaxLineVals() :
            fSpecRec( 0 ),
            fOblique( 0 ) {}

void Init( void )
{ fSpecRec = 0; fMaxLead.Init( scFlowDir( eRomanFlow ) );
  fMaxInkExtents.Set( 0, 0, 0, 0 ); fOblique = 0; }

scSpecRecord* fSpecRec;
scLEADRefData fMaxLead;
scXRect       fMaxInkExtents;
scAngle       fOblique;
};

/* ----- */

enum eBreakEvent {
    start_of_line,
    in_line,
    measure_exceeded,
    end_of_stream_reached
};

typedef eBreakEvent (*BrFunc)( void );

class BreakStruct {
public:
    BreakStruct();
    ~BreakStruct();

    void Init();

    BrFunc* breakMach;
    CandBreak* candBreak;

    // CURRENT BREAK POINT STATE
    CandBreak cB;

    scMemHandle brkLineValsH;
    // a list of max line vals for each spec on the line */
    scMaxLineVals* fMaxLineVals;
    // zero this and make sure it stays that way */
    scMaxLineVals fZeroMaxLineVals;

    CharRecordP gStartRec;

    TypeSpec pspec_;

    scSpecRecord* theSpecRec;

    MicroPoint tmpMinGlue;
    MicroPoint tmpOptGlue;
    MicroPoint tmpMaxGlue;
    GlyphSize letterSpaceAdj;

    MicroPoint originalMeasure;
    MicroPoint desiredMeasure;
    MicroPoint hyphenationZone;

    // length of last line set, for ragged setting */
    MicroPoint lastLineLen;
    GlyphSize justSpace;
    MicroPoint theLineOrg;

    /* space set by character indent */
    MicroPoint charIndent;
    MicroPoint minRelPosition;

    /* we need local values of this in case
     * the spec changes on the line
     */
    MicroPoint brkLeftMargin;
    MicroPoint brkRightMargin;

```

```

MicroPoint      totalTrailingSpace;

long            theLineCount;

Bool            firstGlue;
Bool            firstBox;
Bool            allowHyphens;
Bool            allowJustification;

Bool            fNoStartline;          /* true if previous char was      */
                                         /* starting punctuation          */
MicroPoint      fLastHangable;        /* width of last character that was hangable */
short           numTargetChars;        /* num target chars rubi applied to */

short           lineHyphenated;

/* this the setting for the line based upon
 * the first spec found on the line or a quad
 * character
 */
eTSJust         effectiveRag;

/* if the column has horz flex we
 * fit all the line flush left and
 * then reposition all the lines
 */
eTSJust         colShapeRag;

scColumn        *theBreakColH;

DropCapInfo      dcInfo;
MicroPoint      dcLastBaseline;

/* true if this line contains a drop cap */
Bool            dcSet;

/* we found a character indent char on this line */
Bool            foundCharIndent;

};

class GlobalColumnStruct {
public:
    GlobalColumnStruct()
    {
    }
    ~GlobalColumnStruct()
    {
    }
    TypeSpec      defaultSpec;
    /* this is the current column we are breaking in */
    scColumn*     theActiveColH;
};

/*****

extern BreakStruct      gbrS;
extern GlobalColumnStruct ggcS;
extern scStreamChangeInfo gStreamChangeInfo;

extern Bool            gHiliteSpaces; // hilite trailing spaces at the end of a line

#endif /* _H_SCGLOBDA */

```

File: SCGLOBDA.H

\$Header: /Projects/Toolbox/ct/Scglobda.h 2 5/30/97 8:45a Wmanis \$

Contains: Global data.

Written by: Lucas

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

```
*****/
#ifndef _H_SCGLOBDA
#define _H_SCGLOBDA
```

```
#ifdef SCMACINTOSH
#pragma once
#endif
```

```
#include "sctypes.h"
#include "scselect.h"
#include "scsetjmp.h"
// #include "scvalue.h"
#include "scspcrec.h"
#include "screfdat.h"
#include "scparagr.h"
```

```
*****/
* FOR USE IN THE LINE BREAKER */
```

```
class CandBreak {
public:
    long    breakCount;    /* the count */
    long    startCount;    /* stream count at start of line */
    long    streamCount;   /* stream count from start of this line */
    ushort  wsSpaceCount;  /* # of inter-word spaces at this break */
    ushort  spaceCount;    /* # of glue spaces in interword spaces */
    ushort  trailingSpaces; /* # of trailing spaces */
    ushort  chCount;       /* # of chars */
    ushort  fillSpCount;   /* # of fillspaces we have to patch */
    int     lineVal;       /* offset into leadvals of lead */
    eBreakType breakVal;   /* goodness of break val */
    MicroPoint minGlue;    /* minimum glue */
    MicroPoint optGlue;    /* width of optGlue to this point */
    MicroPoint maxGlue;    /* max glue */
    MicroPoint curBox;     /* width of immovable to this point */
    MicroPoint fHangable;  /* width of hanging character if any */
    CharRecordP theChRec;  /* pointer into stream */
    short    specChanged;  /* spec changed since last candidate */
    TypeSpec spec;         /* spec at this break point */
    scSpecRecord *specRec;
```

```
    CandBreak();
    void Init();
```

```
    CandBreak& operator=( const CandBreak& );
```

```
};
```

```
class scMaxLineVals {
public:
```

File: SCHRECT.C

\$Header: /Projects/Toolbox/ct/SCHRECT.CPP 2 5/30/97 8:45a Wmanis \$

Contains:

This file duplicates in high res rectangles the
'Calculations on Rectangles' described in Inside MAC I-174

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```
*****/
#include "sctypes.h"
```

```
#if defined( _MSC_VER )
#pragma warning(disable:4244) // disable - int conversion
#endif
```

```

===== */
===== */
===== CMUPOINT ===== */
===== */
===== */
```

```
void scMuPoint::FourthToThird( MicroPoint w )
{
    MicroPoint xPrime,
               yPrime;

    scAssert( x != kInvalMP || y != kInvalMP );

    xPrime = y;
    yPrime = w - x;
    x = xPrime;
    y = yPrime;
}
```

```
/* ===== */
```

```
void scMuPoint::ThirdToFourth( MicroPoint w )
{
    MicroPoint xPrime,
               yPrime;

    scAssert( x != kInvalMP || y != kInvalMP );

    xPrime = w - y;
    yPrime = x;
    x = xPrime;
    y = yPrime;
}
```

```
/* ===== */
```

```
/* ===== */
/* ===== */
```

File: MEM.C

\$Header: /Projects/Toolbox/ct/SCMEM.CPP 2 5/30/97 8:45a Wmanis \$

Contains: Memory management routines based on our own heap managers

Written by: Sealy

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/

#include "scmem.h"

#if !useSMARTHEAP

#include <malloc.h>

struct MacHandle {
    const void* fBlock;
    int         fCount;

    MacHandle( scMemHandle ptr ) :
        fBlock( (char*)ptr + sizeof( MacHandle ) ),
        fCount( 0 ) {}

    void* Lock( void ) { scAssert( fCount >= 0 ); fCount++; return (void*)fBlock; }
    void  Unlock( void ) { scAssert( fCount > 0 ); --fCount; }
};

#endif

#include "scexcept.h"
#include <string.h>

#if SCDEBUG > 1
    #include <stdlib.h> // for rand
#endif

/* ***** */
/* ***** */
/* ***** */

#if useSMARTHEAP > 0

static MEM_POOL    hndPool;

static int         numPools;
static scPoolInfo* pools;

/* ***** */

inline MEM_POOL GetHandlePool( void )
{
    return pools[ numPools - 1 ].fPool;
}

/* ***** */

inline MEM_POOL GetPool( size_t size )
{

```



```

    register i;

    for ( i = 0; i < numPools; i++ ) {
        if ( pools[i].fBlockSize && pools[i].fBlockSize == size )
            return pools[i].fPool;
        else
            return pools[i].fPool;
    }
    return 0;
}

/* ===== */

inline MEM_POOL PoolOfPtr( void* ptr )
{
    MEM_POOL_INFO    info;

    if ( MemPoolInfo( 0, ptr, &info ) )
        return info.pool;
    else
        return 0;
}

/* ===== */

inline int CountPools( scPoolInfo infoPools[] )
{
    register i;

    for ( i = 0; infoPools[i++].fBlockSize; )
        ;

    return i;
}

/* ===== */

void MEMInit( scPoolInfo infoPools[] )
{
    register i;

    pools = infoPools;
    numPools = CountPools( pools );

    for ( i = 0; i < numPools; i++ ) {
        if ( pools[i].fBlockSize ) {
            pools[i].fPool = MemPoolInitFS( pools[i].fBlockSize,
                                             1024,
                                             MEM_POOL_DEFAULT );
            raise_if( pools[i].fPool == 0, scERRmem );
        }
        else {
            pools[i].fPool = MemPoolInit( MEM_POOL_DEFAULT );
            raise_if( pools[i].fPool == 0, scERRmem );
        }
    }
}

/* ===== */

#if SCDEBUG > 1

void dbgMemFormatPoolInfo( MEM_POOL_INFO* info )
{
    scChar buf[256];

    SCDebugTrace( 0, scString( "MEM_POOL_INFO\n" ) );

    scStrcpy( buf, scString( "" ) );
    if ( info->type & MEM_FS_BLOCK )
        scStrcat( buf, scString( "MEM_FS_BLOCK " ) );
    if ( info->type & MEM_VAR_MOVEABLE_BLOCK )

```

```

        scStrcat( buf, scString( "MEM_VAR_MOVEABLE_BLOCK " ) );
    if ( info->type & MEM_VAR_FIXED_BLOCK )
        scStrcat( buf, scString( "MEM_VAR_FIXED_BLOCK " ) );
    SCDebugTrace( 0, scString( "MEM_BLOCK_TYPE %s\n" ), buf );

    SCDebugTrace( 0, scString( "pagesize %d\n" ), info->pageSize );
    SCDebugTrace( 0, scString( "floor %lu\n" ), info->floor );
    SCDebugTrace( 0, scString( "ceiling %lu\n" ), info->ceiling );
    SCDebugTrace( 0, scString( "flags 0x%08x\n" ), info->flags );

}
#endif

/* ===== */

void MEMFini()
{
    register    i;

    #if SCDEBUG > 1
        MEM_POOL_INFO    info;
        SCDebugTrace( 0, scString( "\n\nMemFini: BEGIN\n" ) );
    #endif

    #if MEM_DEBUG
        dbgMemSetDefaultErrorOutput( DBGMEM_OUTPUT_CONSOLE, "leakage.out" );
    #endif

    for ( i = 0; i < numPools; i++ ) {
        SCDebugTrace( 0, scString( "Free MemPool - start %d\n" ), i );
        #if SCDEBUG > 1
            MemPoolCheck( pools[i].fPool );
            MemPoolInfo( pools[i].fPool, 0, &info );
            dbgMemFormatPoolInfo( &info );
        #endif
        #if MEM_DEBUG
            scAssert( dbgMemReportLeakage( pools[i].fPool, 1, UINT_MAX ) );
        #endif
        scAssert( MemPoolFree( pools[i].fPool ) ), pools[i].fPool = 0;
        SCDebugTrace( 0, scString( "Free MemPool - end %d\n\n\n" ), i );
    }
    #if SCDEBUG > 1
        SCDebugTrace( 0, scString( "MemFini: DONE\n" ) );
    #endif
}

/* ===== */
#else
/* ===== */

void MEMInit( scPoolInfo [ ] )
{
}

/* ===== */

void MEMFini()
{
}

#endif

/* ===== */
/* ===== */
/* ===== */
#endif SCMACINTOSH

```

```
int          gStartupCompleted;
```

```
// NOTE: To understand this you should be aware of the Macintosh memory
// management as well as the handling of memory in the CApplication class.
// Read the TCL description of the CApplication class and how it handles
// the rainy day fund.
// The stack object CLoanApp tells the application that we can fail this
// memory request. We will assume that all other requests cannot fail.
// That means we must have sufficient memory to service the request.
```

```
// this should really be a CStackObject - unfortunately the chicken/egg
// problem arises because the init of tcExceptContext calls these routines
// and CStackObject relies upon tcExceptContext already existing.
// The reason we would like it to be a stack object is that if we
// throw an exception this would reset the memory requests properly.
// To reset the memory request flags in the application I will
// set them when we ignore the exception at the top of the event loop.
```

```
class CLoanApp {
public:
    CLoanApp();
    ~CLoanApp();
private:
};
```

```
CLoanApp::CLoanApp()
```

```
CLoanApp::~~CLoanApp()
```

```
else
```

```
int          gStartupCompleted = true;
```

```
#define CLoanApp
```

```
#define loanApp
```

```
#endif
```

```
/* ===== */
/* ===== */
/* ===== */
```

```
#if SCDEBUG < 2
```

```
/* ===== */
```

```
void *MEMAllocPtr( ulong sz )
```

```
{
    CLoanApp    loanApp;
    void        *ptr;
```

```
#if useSMARTHEAP
    ptr = MemAllocPtr( GetPool( sz ), sz, 0 );
```

```
#else
    ptr = malloc( sz );
```

```
#endif
```

```
    raise_if( !ptr, scERRmem );
    return ptr;
```

```
}
```

```
/* ===== */
```

```
scMemHandle MEMAllocHnd( ulong sz )
```

```
{
```

```

    CLoanApp    loanApp;
    scMemHandle hnd = 0;

#if useSMARTHEAP
    hnd = MemAlloc( GetHandlePool(), MEM_MOVEABLE | MEM_RESIZEABLE, sz );
#else
    hnd = (scMemHandle)malloc( sizeof( MacHandle ) + sz );

    MacHandle macHandle( hnd );

    *(MacHandle*)hnd = macHandle;
#endif

    raise_if( !hnd, scERRmem );
    return hnd;
}

/* ===== */

//void *MEMAllocObj( ulong size )
//{
//    CLoanApp    loanApp;
//    void        *ptr;
//
//    ptr = GetMemManager().AllocObj( (size_t)size );
//    raise_if( !ptr, scERRmem );
//    return ptr;
//}

/* ===== */

void *MEMDupPtr( void *obj )

    CLoanApp    loanApp;
    void        *ptr;
    ulong       sz = MEMGetSizePtr( obj );

    ptr = MEMAllocPtr( sz );
    raise_if( !ptr, scERRmem );
    SCmemcpy( ptr, obj, sz );
    return ptr;

/* ===== */

scMemHandle MEMDupHnd( scMemHandle obj )

    CLoanApp    loanApp;
    scMemHandle hnd;

#if useSMARTHEAP
    ulong       sz = MemSize( obj );

    hnd = MEMAllocHnd( sz );

    try {
        void*    srcP = MemLock( obj );
        void*    dstP = MemLock( hnd );
        SCmemcpy( dstP, srcP, sz );
    }

    catch( ... ) {
        MemUnlock( hnd );
        MemUnlock( obj );
    }

    MemUnlock( hnd );
    MemUnlock( obj );

#else

    ulong       sz = MEMGetSizePtr( obj );

```

```

hnd = MEMAllocHnd( sz );

try {
    void* srcP = MEMLockHnd( obj );
    void* dstP = MEMLockHnd( hnd );
    SCmemcpy( dstP, srcP, sz );
}

catch( ... ) {
    MEMUnlockHnd( hnd );
    MEMUnlockHnd( obj );
}

MEMUnlockHnd( hnd );
MEMUnlockHnd( obj );

#endif

return hnd;
}

/* ----- */

void *MEMDupObj( void *obj )
{
    CLoanApp    loanApp;
    void        *ptr;
    ulong       sz = MEMGetSizePtr( obj );

    ptr = MEMAllocPtr( sz );
    raise_if( !ptr, scERRmem );
    SCmemcpy( ptr, obj, sz );
    return ptr;
}

/* ----- */

void* MEMResizePtr( void** obj, ulong reqSize )
{
    CLoanApp    loanApp;
    void        *ptr;

    #if useSMARTHEAP
    if ( !*obj )
        ptr = MEMAllocPtr( reqSize );
    else
        ptr = MemReAllocPtr( *obj, reqSize, MEM_RESIZEABLE );
    #else
    if ( !*obj )
        ptr = malloc( reqSize );
    else
        ptr = realloc( *obj, reqSize );
    #endif
    raise_if( !ptr, scERRmem );
    return *obj = ptr;
}

/* ----- */

scMemHandle MEMResizeHnd( scMemHandle obj, ulong reqSize )
{
    CLoanApp    loanApp;

    #if useSMARTHEAP
    if ( !obj )
        obj = MEMAllocHnd( reqSize );
    else
        obj = MemReAlloc( obj, reqSize, MEM_RESIZEABLE );
    #else
    if ( !obj )
        obj = MEMAllocHnd( reqSize );
    else
        obj = (scMemHandle)realloc( obj, reqSize + sizeof( MacHandle ) );
    #endif
}

```

```

    MacHandle macHandle( obj );

    *(MacHandle*)obj = macHandle;
#endif

    return obj;
}

/* ===== */
#endif /* !SCDEBUG */

/* ===== */

void MEMFreePtr( void *obj )
{
    if ( obj == 0 )
        return;

#ifdef useSMARTHEAP
    MemFreePtr( obj );
#else
    free( obj );
#endif
}

/* ===== */

void MEMFreeHnd( scMemHandle obj )
{
    if ( obj == 0 )
        return;

#ifdef useSMARTHEAP
    raise_if( MemLockCount( obj ), scERRmem );
    MemFree( obj );
#else
    free( obj );
#endif
}

/* ===== */

ulong MEMGetSizePtr( const void *obj )
{
    if ( obj == 0 )
        return 0;

#ifdef useSMARTHEAP
    return MemSizePtr( (void*)obj );
#else
    return _msize( (void*)obj );
#endif
}

/* ===== */

ulong MEMGetSizeHnd( scMemHandle obj )
{
    if ( obj == 0 )
        return 0;

#ifdef useSMARTHEAP
    return MemSize( obj );
#else
    return _msize( (void*)obj ) - sizeof( MacHandle );
#endif
}

/* ===== */

void *MEMLockHnd( scMemHandle hnd, int counted )

```

```

{
#ifdef useSMARTHEAP
    return MemLock( hnd );
#else
    MacHandle* mh = (MacHandle*)hnd;
    return mh->Lock();
#endif
}

/* ===== */

void MEMUnlockHnd( scMemHandle hnd, int counted )
{
#ifdef useSMARTHEAP
    MemUnlock( hnd );
#else
    MacHandle* mh = (MacHandle*)hnd;
    mh->Unlock();
#endif
}

/* ===== */

#ifdef SCDEBUG > 1

/* ===== */

void MEMValidate( void *ptr )
{
#ifdef useSMARTHEAP
    MEM_POOL pool = PoolOfPtr( ptr );
    if ( pool ) {
        scAssert( MemPoolCheck( pool ) );
    }
#else
#endif
}

/* ===== */

void memDumpMetrics()
{
#ifdef useSMARTHEAP
#elif useMACHACK
#endif
}

/* ===== */

inline void memRecordTrackInfo( void *ptr, const char *filename, int line )
{
#ifdef MEM_TRACK_ALLOC
    #ifdef useSMARTHEAP
    #else
    #endif
#endif
}

/* ===== */

inline void memRecordTrackInfo( scMemHandle ptr, const char *filename, int line )
{
#ifdef MEM_TRACK_ALLOC
    #ifdef useSMARTHEAP
    #else
    #endif
#endif
}

/* ===== */

```

```

int gRandomFailure; // randomly fail memory allocations

static Boolean RandomFailure()
{
    if ( !gRandomFailure || !gStartUpCompleted )
        return false;

    if ( ( rand() % gRandomFailure ) )
        return false;
    else {
        SCDebugTrace( 0, scString( "RANDOM FAILURE %d\n" ), gRandomFailure );
        return true;
    }
}

/* ===== */

void* MEMAllocPtrDebug( ulong sz, const char *filename, int line )
{
    CLoanApp    loanApp;
    void        *ptr;

    raise_if( RandomFailure(), scERRmem );

#ifdef useSMARTHEAP
    ptr = _dbgMemAllocPtr( GetPool( sz ), sz, 0, filename, line );
#else
    ptr = malloc( sz );
#endif
    raise_if( !ptr, scERRmem );
    memRecordTrackInfo(ptr, filename, line);

    return ptr;

/* ===== */

MacMemHandle MEMAllocHndDebug( ulong sz, const char *filename, int line )
{
    CLoanApp    loanApp;
    scMemHandle hnd;

    raise_if( RandomFailure(), scERRmem );

#ifdef useSMARTHEAP
    hnd = _dbgMemAlloc( GetHandlePool(), MEM_MOVEABLE | MEM_RESIZEABLE, sz, filename, line );
#else
    hnd = (scMemHandle)malloc( sizeof( MacHandle ) + sz );

    MacHandle macHandle( hnd );

    *(MacHandle*)hnd = macHandle;
#endif

    raise_if( !hnd, scERRmem );
    memRecordTrackInfo( hnd, filename, line);

    return hnd;
}

/* ===== */

void* MEMResizePtrDebug( void**      obj,
                        ulong         reqSize,
                        const char*   file,
                        int           line )
{
    CLoanApp    loanApp;
    void        *ptr;

```



```

#if useSMARTHEAP
    if ( !*obj )
        ptr = MEMAllocPtrDebug( reqSize, file, line );
    else
        ptr = _dbgMemReAllocPtr( *obj, reqSize, MEM_RESIZEABLE, file, line );
#else
    ptr = realloc( *obj, reqSize );
#endif

    raise_if( !ptr, scERRmem );
    return *obj = ptr;
}

/* ----- */

scMemHandle MEMResizeHndDebug( scMemHandle  obj,
                               ulong         reqSize,
                               const char*   file,
                               int           line )
{
    CLoanApp    loanApp;

#if useSMARTHEAP
    if ( !obj )
        obj = MEMAllocHndDebug( reqSize, file, line );
    else
        obj = _dbgMemReAlloc( obj, reqSize, MEM_RESIZEABLE, file, line );
#else
    obj = (scMemHandle)realloc( obj, reqSize + sizeof( MacHandle ) );
#endif
    return obj;
}

/* ----- */

void *MEMDupPtrDebug( void *obj, const char *filename, int line )
{
    CLoanApp    loanApp;
    void        *ptr;

    if ( !RandomFailure() ) {
#if useSMARTHEAP
        ulong    sz = MemSizePtr( obj );

        ptr = MEMAllocPtrDebug( sz, filename, line );
        raise_if( !ptr, scERRmem );
        SCmemcpy( ptr, obj, sz );
#else
        ulong    sz = _msize( obj );

        ptr = MEMAllocPtrDebug( sz, filename, line );
        raise_if( !ptr, scERRmem );
        SCmemcpy( ptr, obj, sz );
#endif
    }
    else
        ptr = NULL;
    raise_if( !ptr, scERRmem );
    memRecordTrackInfo( ptr, filename, line );
    return ptr;
}

/* ----- */

scMemHandle MEMDupHndDebug( scMemHandle obj, const char *filename, int line )
{
    CLoanApp    loanApp;
    scMemHandle hnd;

    if ( !RandomFailure() ) {
#if useSMARTHEAP

```

```

        ulong    sz = MemSize( obj );

        hnd = MEMAllocHndDebug( sz, filename, line );

        try {
            void*   srcP = MemLock( obj );
            void*   dstP = MemLock( hnd );
            SCmemcpy( dstP, srcP, sz );
        }

        catch ( ... ) {
            MemUnlock( hnd );
            MemUnlock( obj );
            throw;
        }

        MemUnlock( hnd );
        MemUnlock( obj );

#else

        ulong    sz = _msize( obj ) - sizeof( MacHandle );

        hnd = MEMAllocHndDebug( sz, filename, line );

        try {
            void*   srcP = MEMLockHnd( obj );
            void*   dstP = MEMLockHnd( hnd );
            SCmemcpy( dstP, srcP, sz );
        }

        catch ( ... ) {
            MEMUnlockHnd( hnd );
            MEMUnlockHnd( obj );
        }

        MEMUnlockHnd( hnd );
        MEMUnlockHnd( obj );

#endif
    }
    else
        hnd = NULL;
    raise_if( !hnd, scERRmem );
    memRecordTrackInfo( hnd, filename, line );
    return hnd;

/* ===== */

#endif /* SCDEBUG */

/* ===== */

scAutoUnlock::scAutoUnlock( scMemHandle hnd )
    : fHandle(hnd)
{
    #if useSMARTHEAP
        MemLock( fHandle );
    #else
        MEMLockHnd( fHandle );
    #endif
}

scAutoUnlock::~scAutoUnlock()
{
    #if useSMARTHEAP
        MemUnlock( fHandle );
    #else
        MEMUnlockHnd( fHandle );
    #endif
}

```

```

/* ----- */
#ifdef SCmemset // we are in a 16 bit world
void scFar* scFar scCDecl SCmemset( void scFar* ptr,
                                     int val,
                                     long len )
{
    return _fmemset( ptr, val, (size_t)len );
}

/* ----- */

void scFar* scFar scCDecl SCmemmove( void scFar* dst,
                                     const void scFar* src,
                                     long len )
{
    return _fmemmove( dst, src, (size_t)len );
}

/* ----- */

void scFar* scFar scCDecl SCmemcpy( void scFar* dst,
                                     const void scFar* src,
                                     long len )
{
    return _memcpy( dst, src, (size_t)len );
}

/* ----- */

int scFar scCDecl SCmemcmp( const void scFar* p1,
                            const void scFar* p2,
                            long len )
{
    return _memcmp( p1, p2, (size_t)len );
}

#endif
/* ----- */

```

File: EXCEPT.H

\$Header: /Projects/Toolbox/ct/SCEXCEPT.H 2 5/30/97 8:45a Wmanis \$

Contains: exception code

Written by: Sealy

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/

#ifndef _H_EXCEPT
#define _H_EXCEPT

#include "sctypes.h"

#if SCDEBUG > 1
    #undef new
#endif

/* ===== */

class scException {
public:
    #if SCDEBUG > 1
        scException( status    errCode = scSuccess,
                     const char* file = 0,
                     int       line = 0 ) :
            fFile( file ),
            fLine( line ),
            fCode( errCode ){ SCDebugBreak(); }
    #else
        scException( status    errCode = scSuccess ) :
            fCode( errCode ){ }
    #endif

    status    GetValue(void) const        { return fCode; }

    #if SCDEBUG > 1
        const char*    fFile;
        const int       fLine;
    #endif

private:
    const status    fCode;
};

#if 0

    #if SCDEBUG > 1
        #define raise(err)                throw( scException( err, __FILE__, __LINE__ ) )
        #define raise_if(exp, err)        ((exp) ? (throw( scException( err, __FILE__, __LINE_
        _)),0) : 0)
        #else
            #ifndef MSVCBUG_1A
                #define raise( scerr )                throw( scException( scerr ) )
                #define raise_if(exp, scerr )          ((exp) ? (throw( scException( scerr )),0) :
0)
            #endif
        #endif
    #endif

```

$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}} \right) = \frac{\partial L}{\partial x}$

```

/*-----
File:      chfile.h

$Header: /Projects/Toolbox/ct/SCDBCSDT.H 2      5/30/97 8:45a Wmanis $

Contains:   Class for reading DBCS files.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.
-----*/

```

```

#ifndef _H_SCDBCSDT
#define _H_SCDBCSDT

#include "sctypes.h"

class scDBCSDetector {
public:
    enum eByteType {
        eFirstByte  = -1,
        eOnlyByte,
        eLastByte,
        eMiddleByte
    }; // id's a byte of a multibyte character

    scDBCSDetector( TypeSpec ts );

    void      setDBCS( TypeSpec ts );
    long      StrLen( const char * ) const;

    eByteType ByteType( uchar ch ) const
    {
        return dbcs_ ? shiftjis_[ch] : eOnlyByte;
    }

private:
    Bool      dbcs_;

    static eByteType  shiftjis_[];
};

#endif /* _H_SCDBCSDT */

```

```
if ( fmt && *fmt ) {
    va_start( args, fmt );
    DbgVPrintf( fmt, args );
    va_end( args );
}
```

```
void AssertFailed( const scChar *exp, const char *file, int line )
{
#ifdef SCMACINTOSH
    SCDebugTrace( 0, scString( "(%s,%ld): assert failed: \"%s\"\n" ), file, line, exp );
#else
    SCDebugTrace( 0, scString( "(%s,%d): assert failed: \"%s\"\n" ), file, line, exp );
#endif

    raise( scERRassertFailed );
    // throw( new scException( scERRassertFailed, file, line ) );
}

/* ===== */
```

Variable	Unit	Mean	SD	Min	Max
Age	Years	45.2	12.5	25	65
Gender	Male/Female	50/50	-	-	-
Education	Years	12.8	2.1	8	16
Income	\$/Year	35,000	15,000	15,000	70,000
Health	Good/Bad	60/40	-	-	-
Marital Status	Married/Single	70/30	-	-	-
Occupation	Various	15	5	1	25
Religion	Various	10	3	1	20
Political Affiliation	Various	12	4	1	25
Smoking Status	Smoker/Non-smoker	30/70	-	-	-
Alcohol Consumption	Units/Week	2.5	1.5	0	5
Exercise Frequency	Times/Week	1.5	1.0	0	3
Dietary Habits	Vegetarian/Non-vegetarian	20/80	-	-	-
Stress Level	Low/Medium/High	30/40/30	-	-	-
Family Size	Number of Children	2.0	1.2	0	4
Home Ownership	Owner/Renter	65/35	-	-	-
Travel Frequency	Times/Month	1.0	0.5	0	2
Vehicle Ownership	Yes/No	80/20	-	-	-
Insurance Coverage	Health/Life/Property	90/85/75	-	-	-
Charitable Giving	Amount/Year	500	200	0	1,000
Volunteer Hours	Hours/Month	2.0	1.0	0	4
Political Participation	Voting/Protesting	40/60	-	-	-
Community Involvement	Yes/No	70/30	-	-	-
Life Satisfaction	1-5 Scale	3.5	0.8	1	5
Overall Well-being	1-10 Scale	6.5	1.5	3	10

File: DEBUG.C

SHheader: /Projects/Toolbox/ct/SCDEBUG.CPP 2 5/30/97 8:45a Wmanis \$

Contains: Debugging routines for composition toolkit.

Written by: Sealy

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

```

*****/

#if defined( SCWINDOWS )
#include <windowsx.h>
#else
#include <stdio.h>
#endif

#include <stdarg.h>
#include "scexcept.h"
/* Debugger output/interrupts
 * ===== */
void DbgVPrintf( const scChar*  fmt,
                 va_list      args )
{
#if defined( SCWINDOWS )
    scChar  buf[256];
    int     len;

    wvsprintf( buf, fmt, args );
    len = scStrlen( buf );

    if ( buf[len - 1] == '\n' ) {
        buf[ len - 1 ] = 0;
        scStrcat( buf, scString( "\r\n" ) );
    }

    OutputDebugString( buf );

#elif defined( SCMACINTOSH )
    scChar  buf[256];
    vsprintf( buf, fmt, args );
    fputs( buf, stderr );

#endif
}

/* ===== */

void SCDebugTrace( int level, const scChar* fmt, ... )
{
    extern int scDebugTrace;

    if ( level > scDebugTrace )
        return;

```



```
#include "scexcept.h"
#include <string.h>
```

```

/*-----
File:      charbyte.c

$Header: /Projects/Toolbox/ct/SCDBCSDT.CPP 2      5/30/97 8:45a Wmanis $

Contains:   DBCS code.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

```

/*-----*/

```

```

#include "scdbcsdt.h"
#include "scstcach.h"

/* ----- */
/* ----- */

setDBCSDetector::setDBCSDetector( TypeSpec ts )
: dbcs_(0)
{
    setDBCS( ts );
}

/* ----- */

void setDBCSDetector::setDBCS( TypeSpec ts )
{
    dbcs_ = ts.ptr() ? scCachedStyle::GetCachedStyle( ts ).GetBreakLang() : false;
}

/* ----- */

long setDBCSDetector::StrLen( const char* str ) const
{
    long len = 0;

    for ( ; *str; ) {
        switch( ByteType ( *str++ ) ) {
            case eOnlyByte:
            case eLastByte:
                len++;
                break;
        }
    }
    return len;
}

/* ----- */

setDBCSDetector::eByteType setDBCSDetector::shiftjis[] = {
    eOnlyByte, /* 0x00 */
    eOnlyByte, /* 0x01 */
    eOnlyByte, /* 0x02 */
    eOnlyByte, /* 0x03 */
    eOnlyByte, /* 0x04 */
    eOnlyByte, /* 0x05 */
    eOnlyByte, /* 0x06 */
    eOnlyByte, /* 0x07 */
    eOnlyByte, /* 0x08 */
    eOnlyByte, /* 0x09 */

```

```
eOnlyByte, /* 0x0a */
eOnlyByte, /* 0x0b */
eOnlyByte, /* 0x0c */
eOnlyByte, /* 0x0d */
eOnlyByte, /* 0x0e */
eOnlyByte, /* 0x0f */
eOnlyByte, /* 0x10 */
eOnlyByte, /* 0x11 */
eOnlyByte, /* 0x12 */
eOnlyByte, /* 0x13 */
eOnlyByte, /* 0x14 */
eOnlyByte, /* 0x15 */
eOnlyByte, /* 0x16 */
eOnlyByte, /* 0x17 */
eOnlyByte, /* 0x18 */
eOnlyByte, /* 0x19 */
eOnlyByte, /* 0x1a */
eOnlyByte, /* 0x1b */
eOnlyByte, /* 0x1c */
eOnlyByte, /* 0x1d */
eOnlyByte, /* 0x1e */
eOnlyByte, /* 0x1f */
eOnlyByte, /* 0x20 */
eOnlyByte, /* 0x21 */
eOnlyByte, /* 0x22 */
eOnlyByte, /* 0x23 */
eOnlyByte, /* 0x24 */
eOnlyByte, /* 0x25 */
eOnlyByte, /* 0x26 */
eOnlyByte, /* 0x27 */
eOnlyByte, /* 0x28 */
eOnlyByte, /* 0x29 */
eOnlyByte, /* 0x2a */
eOnlyByte, /* 0x2b */
eOnlyByte, /* 0x2c */
eOnlyByte, /* 0x2d */
eOnlyByte, /* 0x2e */
eOnlyByte, /* 0x2f */
eOnlyByte, /* 0x30 */
eOnlyByte, /* 0x31 */
eOnlyByte, /* 0x32 */
eOnlyByte, /* 0x33 */
eOnlyByte, /* 0x34 */
eOnlyByte, /* 0x35 */
eOnlyByte, /* 0x36 */
eOnlyByte, /* 0x37 */
eOnlyByte, /* 0x38 */
eOnlyByte, /* 0x39 */
eOnlyByte, /* 0x3a */
eOnlyByte, /* 0x3b */
eOnlyByte, /* 0x3c */
eOnlyByte, /* 0x3d */
eOnlyByte, /* 0x3e */
eOnlyByte, /* 0x3f */
eOnlyByte, /* 0x40 */
eOnlyByte, /* 0x41 */
eOnlyByte, /* 0x42 */
eOnlyByte, /* 0x43 */
eOnlyByte, /* 0x44 */
eOnlyByte, /* 0x45 */
eOnlyByte, /* 0x46 */
eOnlyByte, /* 0x47 */
eOnlyByte, /* 0x48 */
eOnlyByte, /* 0x49 */
eOnlyByte, /* 0x4a */
eOnlyByte, /* 0x4b */
eOnlyByte, /* 0x4c */
eOnlyByte, /* 0x4d */
eOnlyByte, /* 0x4e */
eOnlyByte, /* 0x4f */
eOnlyByte, /* 0x50 */
eOnlyByte, /* 0x51 */
eOnlyByte, /* 0x52 */
```

```

eOnlyByte, /* 0x53 */
eOnlyByte, /* 0x54 */
eOnlyByte, /* 0x55 */
eOnlyByte, /* 0x56 */
eOnlyByte, /* 0x57 */
eOnlyByte, /* 0x58 */
eOnlyByte, /* 0x59 */
eOnlyByte, /* 0x5a */
eOnlyByte, /* 0x5b */
eOnlyByte, /* 0x5c */
eOnlyByte, /* 0x5d */
eOnlyByte, /* 0x5e */
eOnlyByte, /* 0x5f */
eOnlyByte, /* 0x60 */
eOnlyByte, /* 0x61 */
eOnlyByte, /* 0x62 */
eOnlyByte, /* 0x63 */
eOnlyByte, /* 0x64 */
eOnlyByte, /* 0x65 */
eOnlyByte, /* 0x66 */
eOnlyByte, /* 0x67 */
eOnlyByte, /* 0x68 */
eOnlyByte, /* 0x69 */
eOnlyByte, /* 0x6a */
eOnlyByte, /* 0x6b */
eOnlyByte, /* 0x6c */
eOnlyByte, /* 0x6d */
eOnlyByte, /* 0x6e */
eOnlyByte, /* 0x6f */
eOnlyByte, /* 0x70 */
eOnlyByte, /* 0x71 */
eOnlyByte, /* 0x72 */
eOnlyByte, /* 0x73 */
eOnlyByte, /* 0x74 */
eOnlyByte, /* 0x75 */
eOnlyByte, /* 0x76 */
eOnlyByte, /* 0x77 */
eOnlyByte, /* 0x78 */
eOnlyByte, /* 0x79 */
eOnlyByte, /* 0x7a */
eOnlyByte, /* 0x7b */
eOnlyByte, /* 0x7c */
eOnlyByte, /* 0x7d */
eOnlyByte, /* 0x7e */
eOnlyByte, /* 0x7f */
eOnlyByte, /* 0x80 */
eFirstByte, /* 0x81 */
eFirstByte, /* 0x82 */
eFirstByte, /* 0x83 */
eFirstByte, /* 0x84 */
eFirstByte, /* 0x85 */
eFirstByte, /* 0x86 */
eFirstByte, /* 0x87 */
eFirstByte, /* 0x88 */
eFirstByte, /* 0x89 */
eFirstByte, /* 0x8a */
eFirstByte, /* 0x8b */
eFirstByte, /* 0x8c */
eFirstByte, /* 0x8d */
eFirstByte, /* 0x8e */
eFirstByte, /* 0x8f */
eFirstByte, /* 0x90 */
eFirstByte, /* 0x91 */
eFirstByte, /* 0x92 */
eFirstByte, /* 0x93 */
eFirstByte, /* 0x94 */
eFirstByte, /* 0x95 */
eFirstByte, /* 0x96 */
eFirstByte, /* 0x97 */
eFirstByte, /* 0x98 */
eFirstByte, /* 0x99 */
eFirstByte, /* 0x9a */
eFirstByte, /* 0x9b */

```

```
eFirstByte, /* 0x9c */
eFirstByte, /* 0x9d */
eFirstByte, /* 0x9e */
eFirstByte, /* 0x9f */
eOnlyByte, /* 0xa0 */
eOnlyByte, /* 0xa1 */
eOnlyByte, /* 0xa2 */
eOnlyByte, /* 0xa3 */
eOnlyByte, /* 0xa4 */
eOnlyByte, /* 0xa5 */
eOnlyByte, /* 0xa6 */
eOnlyByte, /* 0xa7 */
eOnlyByte, /* 0xa8 */
eOnlyByte, /* 0xa9 */
eOnlyByte, /* 0xaa */
eOnlyByte, /* 0xab */
eOnlyByte, /* 0xac */
eOnlyByte, /* 0xad */
eOnlyByte, /* 0xae */
eOnlyByte, /* 0xaf */
eOnlyByte, /* 0xb0 */
eOnlyByte, /* 0xb1 */
eOnlyByte, /* 0xb2 */
eOnlyByte, /* 0xb3 */
eOnlyByte, /* 0xb4 */
eOnlyByte, /* 0xb5 */
eOnlyByte, /* 0xb6 */
eOnlyByte, /* 0xb7 */
eOnlyByte, /* 0xb8 */
eOnlyByte, /* 0xb9 */
eOnlyByte, /* 0xba */
eOnlyByte, /* 0xbb */
eOnlyByte, /* 0xbc */
eOnlyByte, /* 0xbd */
eOnlyByte, /* 0xbe */
eOnlyByte, /* 0xbf */
eOnlyByte, /* 0xc0 */
eOnlyByte, /* 0xc1 */
eOnlyByte, /* 0xc2 */
eOnlyByte, /* 0xc3 */
eOnlyByte, /* 0xc4 */
eOnlyByte, /* 0xc5 */
eOnlyByte, /* 0xc6 */
eOnlyByte, /* 0xc7 */
eOnlyByte, /* 0xc8 */
eOnlyByte, /* 0xc9 */
eOnlyByte, /* 0xca */
eOnlyByte, /* 0xcb */
eOnlyByte, /* 0xcc */
eOnlyByte, /* 0xcd */
eOnlyByte, /* 0xce */
eOnlyByte, /* 0xcf */
eOnlyByte, /* 0xd0 */
eOnlyByte, /* 0xd1 */
eOnlyByte, /* 0xd2 */
eOnlyByte, /* 0xd3 */
eOnlyByte, /* 0xd4 */
eOnlyByte, /* 0xd5 */
eOnlyByte, /* 0xd6 */
eOnlyByte, /* 0xd7 */
eOnlyByte, /* 0xd8 */
eOnlyByte, /* 0xd9 */
eOnlyByte, /* 0xda */
eOnlyByte, /* 0xdb */
eOnlyByte, /* 0xdc */
eOnlyByte, /* 0xdd */
eOnlyByte, /* 0xde */
eOnlyByte, /* 0xdf */
eFirstByte, /* 0xe0 */
eFirstByte, /* 0xe1 */
eFirstByte, /* 0xe2 */
eFirstByte, /* 0xe3 */
eFirstByte, /* 0xe4 */
```

0x9c 0x9d 0x9e 0x9f 0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf 0xb0 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf 0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7 0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf 0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb 0xdc 0xdd 0xde 0xdf 0xe0 0xe1 0xe2 0xe3 0xe4

$$\}:$$
$$\begin{array}{ccccccc} \{1^{(1)}\} & \{2^{(1)}\} & \{3^{(1)}\} & \{4^{(1)}\} & \{5^{(1)}\} & \{6^{(1)}\} & \{7^{(1)}\} \\ \{1^{(2)}\} & \{2^{(2)}\} & \{3^{(2)}\} & \{4^{(2)}\} & \{5^{(2)}\} & \{6^{(2)}\} & \{7^{(2)}\} \\ \{1^{(3)}\} & \{2^{(3)}\} & \{3^{(3)}\} & \{4^{(3)}\} & \{5^{(3)}\} & \{6^{(3)}\} & \{7^{(3)}\} \\ \{1^{(4)}\} & \{2^{(4)}\} & \{3^{(4)}\} & \{4^{(4)}\} & \{5^{(4)}\} & \{6^{(4)}\} & \{7^{(4)}\} \\ \{1^{(5)}\} & \{2^{(5)}\} & \{3^{(5)}\} & \{4^{(5)}\} & \{5^{(5)}\} & \{6^{(5)}\} & \{7^{(5)}\} \\ \{1^{(6)}\} & \{2^{(6)}\} & \{3^{(6)}\} & \{4^{(6)}\} & \{5^{(6)}\} & \{6^{(6)}\} & \{7^{(6)}\} \\ \{1^{(7)}\} & \{2^{(7)}\} & \{3^{(7)}\} & \{4^{(7)}\} & \{5^{(7)}\} & \{6^{(7)}\} & \{7^{(7)}\} \end{array}$$

File: SCPARAGR.H

\$Header: /Projects/Toolbox/ct/SCPARAGR.H 3 5/30/97 8:45a Wmanis \$

Contains: Method/Function interface to class of paragraph

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```
*****/
#ifdef _H_SCPARAGR
#define _H_SCPARAGR
```

```
#ifdef SCMACINTOSH
#pragma once
#endif
```

```
#include "sctbobj.h"
#include "sccharex.h"
#include "scspcrec.h"
#include "scmemarr.h"
```

```
////////////////////////////////////
// FORWARD REFERENCES //
////////////////////////////////////
```

```
#ifdef _RUBI_SUPPORT
class scRubiArray;
#endif
```

```
class scColumn;
class scCOLRefData;
class scSpecRecord;
class scMuPoint;
class scAnnotation;
class scLEADRefData;
class stTextImportExport;
class scTypeSpecList;
class scSpecLocList;
class scTextline;
```

```
/* ***** */
// events that the reformatter returns
```

```
typedef enum eReformatEvents {
    eNoReformat,           // no reformatting was performed
    eNormalReformat,       // normal reformatting event
    eRebreak,              // rebreak the paragraph, probably for widow/orphan control
    eOverflowGeometry,     // more text than columns
    eOverflowContent       // more columns than text
} eRefEvent;
```

```
/* ***** */
```

```
class scStreamChangeInfo {
public:
    // these are the paragraph and offset of character insertion
```



```
// that is used to insure correct update with mono-spaced
// characters
```

```
scStreamChangeInfo( ) :
    fColumn( 0 ),
    fPara( 0 ),
    fOffset( 0 ),
    fLength( 0 ){}

```

```
void Set( scColumn* col, scContUnit* para, long offset, long len ) {
    fColumn    = col,
    fPara      = para,
    fOffset    = offset,
    fLength    = len;
}

```

```
scColumn* GetColumn( void ) const { return fColumn; }
scContUnit* GetPara( void ) const { return fPara; }
long GetOffset( void ) const { return fOffset; }

long GetLength( void ) const { return fLength; }
void SetLength( long len ) { fLength = len; }

```

```
private:
```

```
scColumn* fColumn;
scContUnit* fPara;
long fOffset;
long fLength;
```

```

===== */
class PrevParaData {
public:
    PrevParaData()
    {
        Init();
    }
    void Init( void )
    {
        lastLineH = 0;
        lastSpec.clear();
    }
    scTextline* lastLineH;
    TypeSpec lastSpec;
};

/* ===== */

```

```
class scCharArray : public scHandleArray {
    scDECLARE_RTTI;
public:
    scCharArray() :
        scHandleArray( sizeof( CharRecord ) )
        {
            CharRecord ch( 0, 0 );
            AppendData( (ElementPtr)&ch ); // add null terminator
        }

```

```
virtual int IsEqual( const scObject& ) const;
```

```
UCS2 GetCharAtOffset( long offset ) const
    { return (((CharRecordP)GetMem()) + offset )->character; }
```

```
void RemoveBetweenOffsets( long startOffset, long endOffset );
```

```
// copy the contents from startOffset to endOffset into the
// arg scCharArray
```

```
void Copy( scCharArray&, long startOffset, long endOffset ) const;
```

```

// copy the contents from startOffset to endOffset to the
// arg scCharArray and then remove them
void Cut( scCharArray&, long, long );

// paste the contents of the arg scCharArray into the character array
// at the indicated array
void Paste( scCharArray&, long startOffset );

int FindString( const stUnivString&, const SearchState&, int32, int32, int32& );
int ReplaceToken( const stUnivString&, int32, int32& );
int GetToken( stUnivString&, int32, int32 ) const;

void Insert( const CharRecordP, long, long );
void Insert( const UCS2*, long, long );
int Insert( const stUnivString&, int32, int32 );

void CopyChars( UCS2*, long, long );

// transform the indicated characters using the type of
// transformation passed in, ususally for making
// alternate characters
void Transform( long startOffset,
               long endOffset,
               eChTranType trans,
               int numChars );

void Retabulate( scSpecRun& specRun,
               long start,
               long end,
               TypeSpec changedSpec,
               long charSize );

void RepairText( scSpecRun&,
               long offset1,
               long offset2 );

void SelectWord( long offset,
               long& startWord,
               long& endWord );

#ifdef _RUBI_SUPPORT
void CharInsert( long&,
               scSpecRun&,
               scRubiArray*,
               long,
               scKeyRecord&,
               Bool,
               TypeSpec );
#else
void CharInsert( long&,
               scSpecRun&,
               long,
               scKeyRecord&,
               Bool,
               TypeSpec );
#endif

void WordSpaceInfo( long, MicroPoint& );

void CharInfo( scSpecRun&,
               long,
               UCS2&,
               ulong&,
               MicroPoint&,
               TypeSpec&,
               eUnitType& );

long ReadText( scSpecRun&,
               APPCtxPtr ctxPtr,
               IOFuncPtr readFunc,
               int charset = 0 );

```

```

void      WriteText( scSpecRun&,
                    Bool,
                    APPCtxPtr   ctxPtr,
                    IOFuncPtr   writeFunc,
                    int          charset = 0 );

long      ReadAPPTText( scSpecRun&, stTextImportExport& );
void      WriteAPPTText( scSpecRun&, stTextImportExport& );

long      GetContentSize( void ) const
        {
            return fNumItems - 1;
        }

void      SetContentSize( long );
long      ExternalSize( void ) const;

void      Read( APPCtxPtr, IOFuncPtr );
void      Write( APPCtxPtr, IOFuncPtr );

virtual ElementPtr Lock( void );
virtual void      Unlock( void );
void      Validate( void ) const;

private:
void      CopyChars( CharRecordP, long, long );

#ifdef _RUBI_SUPPORT
void      DoBackSpace( long&,
                    long&,
                    scSpecRun&,
                    scRubiArray*,
                    long,
                    scKeyRecord&,
                    Bool );

void      DoForwardDelete( long&,
                    long&,
                    scSpecRun&,
                    scRubiArray*,
                    long,
                    scKeyRecord&,
                    Bool );

void      DoDiscHyphen( long&,
                    long&,
                    scSpecRun&,
                    scRubiArray*,
                    long,
                    scKeyRecord&,
                    Bool );

void      DoFixSpace( long&,
                    long&,
                    scSpecRun&,
                    scRubiArray*,
                    long,
                    scKeyRecord&,
                    Bool );

void      DoCharacter( long&,
                    long&,
                    scSpecRun&,
                    scRubiArray*,
                    long,
                    scKeyRecord&,
                    Bool );

#else
void      DoBackSpace( long&,
                    long&,
                    scSpecRun&,

```

```

scKeyRecord&,
Bool );

```

```

void DoForwardDelete( long&,
                      long&,
                      scSpecRun&,
                      long,
                      scKeyRecord&,
                      Bool );

```

```

void DoDiscHyphen( long&,
                  long&,
                  scSpecRun&,
                  long,
                  scKeyRecord&,
                  Bool );

```

```

void DoFixSpace( long&,
                long&,
                scSpecRun&,
                long,
                scKeyRecord&,
                Bool );

```

```

void DoCharacter( long&,
                 long&,
                 scSpecRun&,
                 long,
                 scKeyRecord&,
                 Bool );

```

```

#endif

```

```

/* ===== */

```

```

long TXTStartWord( CharRecordP, long, int eliminateLeadingSpaces );
long TXTEndWord( CharRecordP, long );
long TXTStartSelectableWord( CharRecordP, long );
long TXTEndSelectableWord( CharRecordP, long );

```

```

MicroPoint UnivStringWidth( stUnivString&, MicroPoint[], TypeSpec& );

```

```

#ifdef jis4051

```

```

Bool TXTSameRenMoji( CharRecordP start, CharRecordP ch1, CharRecordP ch2 );

```

```

#else

```

```

inline Bool TXTSameRenMoji( CharRecordP, CharRecordP, CharRecordP )

```

```

{

```

```

    return false;

```

```

}

```

```

#endif

```

```

/* ===== */
/* ===== */
/* ===== */

```

```

class scContUnit : public scTBObj {

```

```

    scDECLARE_RTTI;

```

```

public:

```

```

    // use this to allocate new content units where the content unit

```

```

    // has been overridden on the outside.

```

```

static scContUnit* Allocate( TypeSpec& spec,
                           scContUnit* cu = 0,
                           long ct = 0 );

```

```

    scContUnit();

```

```

    scContUnit( TypeSpec& spec,
               scContUnit* cu = 0,
               long ct = 0 );

```

File: pfileio.c

\$Header: /Projects/Toolbox/ct/SCFILEIO.CPP 2 5/30/97 8:45a Wmanis \$

Contains: Implementation of independent byte order code.

Written by: Coletti

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/

#include "scfileio.h"
#include "scexcept.h"
#include "scmem.h"
#include <string.h>

#if 0
#include "cvtfloat.h"
#endif
// the string that goes into the header
static char *byteOrderStr[] = {
    "",
    "Intel86",
    "Motor68",
    NULL
}

----- */
/* ----- */

// Intel to Motorola
#define SC_I2M_MKWORD(p) (((ushort) ((p)[1]) << 8 ) | ( (p)[0] ))

// Motorola to Intel
#define SC_M2I_MKWORD(p) (((ushort) ((p)[0]) << 8 ) | ( (p)[1] ))

// Intel to Motorola
#define SC_I2M_MKLONG(p) \
    ((long)(ulong)SC_I2M_MKWORD(p) | (((long)(ulong)SC_I2M_MKWORD((p)+2)) << 16))

// Motorola to Intel
#define SC_M2I_MKLONG(p) \
    ((long)(ulong)SC_M2I_MKWORD((p)+2) | (((long)(ulong)SC_M2I_MKWORD(p)) << 16))

/* ----- */

#if defined( SCWINDOWS ) && !defined( _X86_ )
#define _X86_ 1
#endif

#if defined( _X86_ )

static int localByteOrder = kIntelOrder;

#define SCPIO_MKWORD SC_M2I_MKWORD
#define SCPIO_MKLONG SC_M2I_MKLONG

#elif defined( SCMACINTOSH )

```

```

static short localByteOrder = kMotorolaOrder;

#define SCPIO_MKWORD      SC_I2M_MKWORD
#define SCPIO_MKLONG      SC_I2M_MKLONG

#endif

#ifndef SCPIO_MKWORD
#error "A Processor architecture needs to be defined"
#endif

/* ----- */
/* ----- */
// code for creating the header

uchar*      BufSet_byteorder( uchar pbuf[] )
{
    SCmemset( pbuf, 0, sizeof( ByteOrderStr ) );

    strcpy( (char*)pbuf, byteOrderStr[localByteOrder] );

    return pbuf + sizeof( ByteOrderStr );
}

/* ----- */
// code for extracting the header

const uchar*      BufGet_byteorder( const uchar  pbuf[],
                                     short*       byteOrder )

{
    if ( !strcmp( (char *)pbuf, byteOrderStr[kMotorolaOrder] ) )
        *byteOrder = kMotorolaOrder;
    else if ( !strcmp( (char *)pbuf, byteOrderStr[kIntelOrder] ) )
        *byteOrder = kIntelOrder;
    else
        *byteOrder = kNoOrder;
    return pbuf + sizeof( ByteOrderStr );
}

/* ----- */

uchar*      BufSet_char( uchar*      dstbuf,
                        const uchar*  srcbuf,
                        size_t        bytes,
                        eByteOrder    )

{
    SCmemmove( dstbuf, srcbuf, bytes );
    return dstbuf + bytes;
}

/* ----- */

const uchar*      BufGet_char( const uchar*  srcbuf,
                               uchar*        dstbuf,
                               size_t        bytes,
                               eByteOrder    )

{
    SCmemmove( dstbuf, srcbuf, bytes );
    return srcbuf + bytes;
}

/* ----- */
// write out a short to a byte buffer

uchar*      BufSet_short( uchar  pbuf[2],
                          ushort  s,
                          eByteOrder  desiredByteOrder )
{
    if ( desiredByteOrder != localByteOrder ) {
        switch ( desiredByteOrder ) {
            case kMotorolaOrder:

```

```

        *pbuf = (uchar) I2M_MKWORD((uchar*)&s);
        break;

    case kIntelOrder:
        *pbuf = (uchar) SC_M2I_MKWORD((uchar*)&s);
        break;

    default:
        *(ushort*)pbuf = s;
    }
}
else
    *(ushort*)pbuf = s;

return pbuf + sizeof( ushort );
}

/* ----- */
// read out a short from a byte buffer

const uchar*   BufGet_short( const uchar  abuf[2],
                             ushort&      s,
                             eByteOrder    byteOrder )
{
    if ( localByteOrder != byteOrder )
        s = (ushort)SCPIO_MKWORD(abuf);
    else
        s = *(ushort*)abuf;

    return abuf+2;
}

/* ----- */
// write out a long to a byte buffer

uchar*         BufSet_long( uchar      pbuf[4],
                             ulong      l,
                             eByteOrder  desiredByteOrder )
{
    if ( desiredByteOrder != localByteOrder ) {
        switch ( desiredByteOrder ) {
            case kMotorolaOrder:
                *(ulong*)pbuf = SC_I2M_MKLONG((uchar*)&l);
                break;

            case kIntelOrder:
                *(ulong*)pbuf = SC_M2I_MKLONG((uchar*)&l);
                break;

            default:
                *((ulong*)pbuf) = l;
        }
    }
    else
        *((ulong*)pbuf) = l;

    return pbuf + sizeof( ulong );
}

/* ----- */
// read out a long from a byte buffer */

const uchar*   BufGet_long( const uchar  abuf[4],
                             ulong&      l,
                             eByteOrder    byteOrder )
{
    if ( localByteOrder != byteOrder )
        l = SCPIO_MKLONG(abuf);
    else
        l = *(ulong*)abuf;
}

```

```

    return abuf+4;
}

/* ----- */

static const uchar* BytesToIntelDouble( const REALBUF    rbuf,
                                         REAL&           r )
{
    uchar    *ptr = (uchar *)&r;

    switch( sizeof( REAL ) ) {
        case 10:
            ptr[9] = rbuf[2];
            ptr[8] = rbuf[3];
            break;
        case 8:
            break;
        default:
            scAssert( 0 );
            break;
    }

    ptr[7] = rbuf[4];
    ptr[6] = rbuf[5];
    ptr[5] = rbuf[6];
    ptr[4] = rbuf[7];
    ptr[3] = rbuf[8];
    ptr[2] = rbuf[9];
    ptr[1] = rbuf[10];
    ptr[0] = rbuf[11];

    return rbuf + 12;
}

/* ----- */

static uchar* IntelDoubleToBytes( REALBUF    rbuf,
                                  REAL         r )
{
    uchar    *ptr = (uchar *)&r;

    switch( sizeof( REAL ) ) {
        case 10:
            rbuf[0] = ptr[9];
            rbuf[1] = ptr[8];
            rbuf[2] = ptr[9];
            rbuf[3] = ptr[8];
            break;

        case 8:
            rbuf[0] = ptr[7];
            rbuf[1] = ptr[6];
            rbuf[2] = ptr[7];
            rbuf[3] = ptr[6];
            break;

        default:
            scAssert( 0 );
            break;
    }

    rbuf[4] = ptr[7];
    rbuf[5] = ptr[6];
    rbuf[6] = ptr[5];
    rbuf[7] = ptr[4];
    rbuf[8] = ptr[3];
    rbuf[9] = ptr[2];
    rbuf[10] = ptr[1];
    rbuf[11] = ptr[0];

    return rbuf + 12;
}

```



```

/* ----- */

static const uchar* BytesToMotorolaDouble( const REALBUF  rbuf,
                                           REAL&          r )
{
    uchar  *ptr = (uchar *)&r;

    switch( sizeof( REAL ) ) {
        case 12:
            SCmemcpy( ptr, rbuf, sizeof( REAL ) );
            break;
        case 10:
            ptr[0] = rbuf[0];
            ptr[1] = rbuf[1];
            ptr[2] = rbuf[4];
            ptr[3] = rbuf[5];
            ptr[4] = rbuf[6];
            ptr[5] = rbuf[7];
            ptr[6] = rbuf[8];
            ptr[7] = rbuf[9];
            ptr[8] = rbuf[10];
            ptr[9] = rbuf[11];
            break;
        case 8:
            #if 0
                // Convert extended representation to 64 bit IEEE format
                {
                    long extendedRep[3];
                    SCmemcpy( extendedRep, rbuf, 12 );
                    CvtFloat96To64( r, extendedRep );
                }
            else
                raise( scERRnotImplemented );
            #endif
            break;
        default:
            scAssert( 0 );
            break;
    }

    return rbuf + 12;
}

/* ----- */

static uchar*  MotorolaDoubleToBytes( REALBUF  rbuf,
                                      REAL      r )
{
    uchar  *ptr = (uchar *)&r;

    switch( sizeof( REAL ) ) {
        case 12:
            SCmemcpy( rbuf, ptr, sizeof( REAL ) );
            break;
        case 10:
            rbuf[0] = ptr[0];
            rbuf[1] = ptr[1];
            rbuf[2] = ptr[0];
            rbuf[3] = ptr[1];
            rbuf[4] = ptr[2];
            rbuf[5] = ptr[3];
            rbuf[6] = ptr[4];
            rbuf[7] = ptr[5];
            rbuf[8] = ptr[6];
            rbuf[9] = ptr[7];
            rbuf[10] = ptr[8];
            rbuf[11] = ptr[9];
            break;
        case 8:
            #if 0
                // Convert 64 bit representation to extended
                {

```

```

        long extendedRep[3];
        CvtFloat64ToLong( extendedRep, &r );
        SCmemcpy( rbuf, extendedRep, 12 );
    }

#else
        raise( scERRnotImplemented );
#endif

        break;

    default:
        scAssert( 0 );
        break;
}

return rbuf + 12;
}

/* ----- */

/* write out a REAL to a byte buffer */

uchar*    BufSet_REAL( uchar    rbuf[12],
                       REAL      d,
                       eByteOrder )
{
    switch ( localByteOrder ) {

        case kIntelOrder:
            // convert intel long double to bytes
            IntelDoubleToBytes( rbuf, d );
            break;

        case kMotorolaOrder:
            // convert motorla long double to bytes
            MotorolaDoubleToBytes( rbuf, d );
            break;

        default:
            scAssert( 0 );
            break;
    }

    return rbuf + 12;
}

/* ----- */

/* read in a REAL from a byte buffer */

const uchar*    BufGet_REAL( const uchar    rbuf[12],
                             REAL&          r,
                             eByteOrder )
{
    switch ( localByteOrder ) {

        case kMotorolaOrder:
            BytesToMotorolaDouble( (uchar *)rbuf, r );
            break;

        case kIntelOrder:
            BytesToIntelDouble( (uchar *)rbuf, r );
            break;

        default:
            break;
    }

    return rbuf + 12;
}

/* ----- */
/* ----- */
// write a quick long

void ReadLong( long&          val,
               APPCtxPtr      ctxPtr,
               IOFuncPtr      readFunc,

```

File: SCCTYPE.H

\$Header: /Projects/Toolbox/ct/SCCTYPE.H 2 5/30/97 8:45a Wmanis \$

Contains: Character types.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication
and does not constitute an admission or acknowledgment that publication
has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary
and confidential property of Stonehand Inc.

```

*****/

#ifndef _H_SCCTYPE
#define _H_SCCTYPE

#ifndef _H_SCTYPES
#include "sctypes.h"
#endif

#define sc_ASCII          0x0001
#define sc_SPACE          0x0002
#define sc_PUNC           0x0004
#define sc_DIGIT          0x0008
#define sc_ALPHA          0x0010
#define sc_ACCENT         0x0020
#define sc_SYMBOL         0x0040
#define sc_LOCASE         0x0080
#define sc_UPCASE         0x0100
#define sc_LIGATR         0x0200

extern unsigned short sc_CharType[];

/* for now we assume everything above 255 is alpha, with release of kanji and other
   versions this will change
*/
#define CTIsAlpha(ch)      ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_ALPHA) ) : true )
#define CTIsSelectable(ch) ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_ALPHA|sc_DIGIT) ) : true )
#define CTIsDigit(ch)     ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_DIGIT) ) : false )
#define CTIsPunc(ch)      ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_PUNC) ) : false )
#define CTIsUpperCase(ch) ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_UPCASE) ) : false )
#define CTIsLowerCase(ch) ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_LOCASE) ) : false )
#define CTIsSpace(ch)     ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_SPACE) ) : false )
#define CTIsSymbol(ch)    ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_SYMBOL) ) : false )
#define CTIsVisible(ch)   ( (ch) < 256 ? ( sc_CharType[(ch)+1] & (sc_ALPHA|sc_DIGIT|sc_ACCENT|sc_P
UNC|sc_SYMBOL) ) : true )

#define CTIsDropCapable(ch) ( CTIsVisible( ch ) && !CTIsSpace( ch ) )

Bool      CTStoreAll( UCS2 );
Bool      CTIsFracBar( UCS2 );
UCS2      CTToLower( UCS2 );
UCS2      CTToUpper( UCS2 );
UCS2      CTToggleCase( UCS2 );

#endif /* _H_SCCTYPE */

```

```

/* ===== C H T ===== */
/* ===== */
/* ===== */

#if SCDEBUG > 1

scChar* scXRect::DebugStr( scChar* buf, int factor ) const
{
    #if defined(SCWINDOWS)
        wsprintf( buf, scString( "(%d, %d, %d, %d)" ), x1 / factor, y1 / factor, x2 / factor, y2 / factor );
    #else
        sprintf( buf, scString( "(%d, %d, %d, %d)" ), x1 / factor, y1 / factor, x2 / factor, y2 / factor );
    #endif
    return buf;
}

#endif

/* ===== */

Bool scXRect::Valid( eCoordSystem coordSys ) const
{
    switch ( coordSys ) {
        case eFirstQuad:
            return x1 <= x2 && y1 >= y2;
        case eSecondQuad:
            return x1 >= x2 && y1 >= y2;
        case eThirdQuad:
            return x1 >= x2 && y1 <= y2;
        case eFourthQuad:
            return x1 <= x2 && y1 <= y2;
    }
    return false;
}

/* ===== */

void scXRect::Scale( TenThousandth factor )
{
    #if SCDEBUG>2
        scAssert( Valid() );
    #endif

    x1 = scRoundMP( (REAL)x1 * factor / 10000.0 );
    x2 = scRoundMP( (REAL)x2 * factor / 10000.0 );
    y1 = scRoundMP( (REAL)y1 * factor / 10000.0 );
    y2 = scRoundMP( (REAL)y2 * factor / 10000.0 );
}

/* ===== */

void scXRect::Scale( REAL factor )
{
    #if SCDEBUG>2
        scAssert( Valid() );
    #endif

    x1 = scRoundMP( (REAL)x1 * factor );
    x2 = scRoundMP( (REAL)x2 * factor );
    y1 = scRoundMP( (REAL)y1 * factor );
    y2 = scRoundMP( (REAL)y2 * factor );
}

/* ===== */

void scXRect::FourthToThird( MicroPoint w )
{
    scMuPoint    pt1,
                pt2;

```

```

#if SCDEBUG>2
    scAssert( Valid() );
#endif

    pt1.x = x2;
    pt1.y = y1;

    pt1.FourthToThird( w );

    pt2.x = x1;
    pt2.y = y2;

    pt2.FourthToThird( w );

    x1 = pt1.x;
    y1 = pt1.y;

    x2 = pt2.x;
    y2 = pt2.y;

#if SCDEBUG>2
    scAssert( Valid() );
#endif
}

/* ===== */

void scXRect::ThirdToFourth( MicroPoint w )
{
    scMuPoint    pt1,
                 pt2;

#if SCDEBUG>2
    scAssert( Valid() );
#endif

    pt1.x = x1;
    pt1.y = y2;

    pt1.ThirdToFourth( w );

    pt2.x = x2;
    pt2.y = y1;

    pt2.ThirdToFourth( w );

    x1 = pt1.x;
    y1 = pt1.y;

    x2 = pt2.x;
    y2 = pt2.y;

#if SCDEBUG>2
    scAssert( Valid() );
#endif
}

/* ===== */

void scXRect::FirstToFourth( MicroPoint d )
{
    #if SCDEBUG>2
        scAssert( Valid() );
    #endif

    y1 = -y1;
    y2 = -y2;

    #if SCDEBUG>2
        scAssert( Valid() );
    #endif
}

```

```

/* ----- */

void scXRect::FourthToFirst( MicroPoint d )
{
    #if SCDEBUG>2
        scAssert( Valid() );
    #endif

    y1 -= d;
    y2 -= d;

    #if SCDEBUG>2
        scAssert( Valid() );
    #endif
}

/* ----- */

/* ----- */
/* ----- */
/* ----- CRLURECT ----- */
/* ----- */
/* ----- */

scRLURect::scRLURect( )
{
    // in an attempt to insure that we can freely convert
    // back and forth between these we do the following text
    scAssert( sizeof( scRLURect ) == sizeof( RLU ) * 4 );

    Invalidate();
}

/* ----- */

scRLURect::scRLURect( const scRLURect& rlurect )
{
    rluLeft      = rlurect.rluLeft;
    rluTop       = rlurect.rluTop;
    rluRight     = rlurect.rluRight;
    rluBottom    = rlurect.rluBottom;
}

/* ----- */

void scRLURect::Set( RLU left, RLU top, RLU right, RLU bottom )
{
    rluLeft      = left;
    rluTop       = top;
    rluRight     = right;
    rluBottom    = bottom;
}

/* ----- */

Bool    scRLURect::Valid( eCoordSystem coordSys ) const
{
    switch ( coordSys ) {
        case eFirstQuad:
            return rluLeft <= rluRight && rluTop >= rluBottom;
        case eSecondQuad:
            return rluLeft >= rluRight && rluTop >= rluBottom;
        case eThirdQuad:
            return rluLeft >= rluRight && rluTop <= rluBottom;
        case eFourthQuad:
            return rluLeft <= rluRight && rluTop <= rluBottom;
    }
    return false;
}

/* ----- */

```

```

void scRLURect::Invalidate( )
{
    Set( SHRT_MAX, SHRT_MAX, SHRT_MIN, SHRT_MIN );
}

/* ===== */

void scRLURect::Translate( RLU h, RLU v )
{
    rluRight    = rluRight  + h;
    rluLeft     = rluLeft   + h;
    rluTop      = rluTop    + v;
    rluBottom   = rluBottom + v;
}

/* ===== */

void scRLURect::FirstToFourth( RLU )
{
    rluTop      = -rluTop;
    rluBottom   = -rluBottom;
}

/* ===== */

void scRLURect::FourthToFirst( RLU )
{
    rluTop      = -rluTop;
    rluBottom   = -rluBottom;
}

/* ===== */

void scRLURect::RLURomanBaseLineToCenter( void )
{
    rluRight    = (rluRight - rluLeft)/2;
    rluLeft     = 0 - rluRight;

    //use bottom as temp variable to save height
    rluBottom   = rluTop - rluBottom;
    rluTop      = scBaseRLUsystem - ( rluTop + RLU_BASEfmBottom);
    rluBottom   = rluTop + rluBottom; //Bottom has character height
}

/* ===== */

void scRLURect::RLURomanBaseLineToLeft( void )
{
    rluRight    = (rluRight - rluLeft);
    rluLeft     = 0;

    //use bottom as temp variable to save height
    rluBottom   = rluTop - rluBottom;
    rluTop      = scBaseRLUsystem - ( rluTop + RLU_BASEfmBottom);
    rluBottom   = rluTop + rluBottom; //Bottom has character height
}

/* ===== */

void scRLURect::RLURomanBaseLineToRight( void )
{
    rluLeft     = 0 - (rluRight - rluLeft);
    rluRight    = 0;
    rluBottom   = rluTop - rluBottom; //use bottom as temp variable to save height
    rluTop      = scBaseRLUsystem - ( rluTop + RLU_BASEfmBottom);
    rluBottom   = rluTop + rluBottom; //Bottom has character height
}

/* ===== */

void scRLURect::RLURomanBaseLineToTop( void )
{

```

```

    Translate( 0, -RLU_BASEfmTop );
}

/* ===== */

void    scRLURect::RLURomanBaseLineToMiddle( void )
{
    Translate( 0, -RLU_BASEfmTop/2 );
}

/* ===== */

void    scRLURect::RLURomanBaseLineToBottom( void )
{
    Translate( 0, RLU_BASEfmBottom );
}

/* ===== */
#ifdef 0
void RectTest( )
{
    scXRect xrect( 100, 100, 200, 200 );
    scMuPoint pt1;

    scMuPoint pt2;

    pt1.x = 20;
    pt1.y = 80;

    pt2 = pt1;

    xrect.FourthToThird( 1000 );
    xrect.ThirdToFourth( 1000 );

    pt1.FourthToThird( 1000 );
    pt1.ThirdToFourth( 1000 );

    pt1.FourthToThird( 100 );
    pt1.ThirdToFourth( 100 );

    pt1.FourthToThird( 200 );
    pt1.ThirdToFourth( 200 );

    pt1.FourthToThird( 500 );
    pt1.ThirdToFourth( 500 );

    scAssert( pt1 == pt2 );
}
#endif
/* ===== */

```



```

        eByteOrder      Order )
{
    uchar buf[4];
    raise_if( (*readFunc)( ctxPtr, buf, 4 ) != 4, scERRfile );
    ulong uval;
    BufGet_long( buf, uval, byteOrder );
    val = (long)uval;
}

/* ----- */
// write a quick long

void WriteLong( long          val,
                APPCtxPtr     ctxPtr,
                IOFuncPtr     writeFunc,
                eByteOrder     byteOrder )
{
    uchar buf[4];
    BufSet_long( buf, val, byteOrder );
    raise_if( (*writeFunc)( ctxPtr, buf, 4 ) != 4, scERRfile );
}

/* ----- */

void ReadBytes( uchar*       buf,
                APPCtxPtr     ctxPtr,
                IOFuncPtr     readFunc,
                long          numbytes )
{
    raise_if( (*readFunc)( ctxPtr, buf, numbytes ) != numbytes, scERRfile );
}

/* ----- */

void WriteBytes( const uchar* buf,
                APPCtxPtr     ctxPtr,
                IOFuncPtr     writeFunc,
                long          numbytes )
{
    raise_if( (*writeFunc)( ctxPtr, (void*)buf, numbytes ) != numbytes, scERRfile );
}

/* ----- */

```

File: SC-CharMap.c

\$Header: /Projects/Toolbox/ct/SC_CHMAP.CPP 6 5/30/97 8:45a Wmanis \$

Contains: Character mapping between client and toolbox.

Written by: Manis

Copyright (c) 1989-94 Stonehand Inc., of Cambridge, MA.
All rights reserved.

This notice is intended as a precaution against inadvertent publication and does not constitute an admission or acknowledgment that publication has occurred or constitute a waiver of confidentiality.

Composition Toolbox software is the proprietary and confidential property of Stonehand Inc.

to turn off any of the functions in this module define one or more of the following values in SCAPPTypes.h, they will turn off the appropriate function.

```
noCMctToAPP
noCMappToCT
noCMmakeKeyRecordTwo
noCMcontent
```

```
#include "sccharex.h"
```

```
#define CTL( ch ) ( (ch) - '@' )
```

```
/* this provides character mapping between Mac-keyboard/application
 * to the Composition Toolboxt
 */
```

```
#ifndef noCMappToCT
```

```
UCS2 CMappToCT( UCS2 ch )
```

```
{
    switch ( ch ) {
        case 0x0D: return scParaSplit;          /* mac enter */
        case 0x08: return scBackSpace;          /* mac delete */
        case 0x09: return scTabSpace;           /* mac tab */
        case 0x0a: return scHardReturn;         /* mac return */

        default: return (UCS2)ch;
    }
}
```

```
#endif /* noCMappToCT */
```

```
/* provides Composition Toolboxt to application mapping,
 * used pre AppDrawString for rationalize character mapping,
 * used to control characters passed thru, typically used
 * to control things like show invisibles, may be used for
 * other types of character conversion depending on output device
 */
```

```
#ifndef noCMctToAPP
```

```
UCS2 CMctToAPP( UCS2 ch )
```

```
{
    switch ( ch ) {
        default:
```

```

        return ch;
    case scNoBreakHyph:
        return '-';

#if 1
    case scTabSpace:
        return 0;
    case scParaEnd:
        return 0;
    case scEndStream:
        return 0;
    case scHardReturn:
        return 0;
#else
    case scTabSpace:
        return 0x00bb;
    case scParaEnd:
        return 0x00b6;
    case scEndStream:
        return 0x00a5;
    case scHardReturn:
        return 'H';
#endif

    case scEmSpace:
    case scEnSpace:
    case scFigureSpace:
    case scThinSpace:
    case scFixRelSpace:
    case scFixAbsSpace:
    case scFillSpace:
    case scVertTab:
    case scNoBreakSpace:
    case scQuadCenter:
    case scQuadLeft:
    case scQuadRight:
    case scQuadJustify:
    case scEmptySpace:
        return ' ';
}

#endif /* noCMctToAPP */

/* *****
 * defines whether keyboard input changes model & selection or selection
 * only. Called from within Composition Toolbox prior to keyboard input
 * to determine what is about to happen
 */

#ifndef noCMcontent

int CMcontent( UCS2 ch )
{
    switch ( ch ) {
        case scBackSpace:
        case scForwardDelete:
            return -1;

        default:
            return 1;

        case scLeftArrow:
        case scRightArrow:
        case scUpArrow:
        case scDownArrow:
            return false;
    }
}

#endif /* noCMcontent */

/* *****

```

```

        {
            return p_ == 0;
        }
    int operator!() const
    {
        return p_ == 0;
    }
    int operator==( const ConstRefCountPtr<T>& p ) const
    {
        return p_ == p.p_;
    }
    int operator!=( const ConstRefCountPtr<T>& p ) const
    {
        return p_ != p.p_;
    }
    int operator==( const T* p ) const
    {
        return p_ == p;
    }
    int operator!=( const T* p ) const
    {
        return p_ != p;
    }

protected:
    T* p_;
};

```

```

template<class T>
class RefCountPtr : public ConstRefCountPtr<T> {
public:
    RefCountPtr( T* ptr = 0 ) :
        ConstRefCountPtr<T>( ptr ){ }

    ~RefCountPtr() { }

    T* ptr() const
    {
        return p_;
    }

    T* operator->() const
    {
        return p_;
    }

    T& operator*() const
    {
        return *p_;
    }

    void exch(RefCountPtr<T> &p)
    {
        T* tmp = p.p_;
        p.p_ = p_;
        p_ = tmp;
    }
};

#endif

```

```
//
// Copyright (c) 1996, Stonehenge Inc. All rights reserved.
//
```

```
#ifndef _H_REFCNT
#define _H_REFCNT
```

```
    // a base class for reference counting
```

```
#ifdef _DEBUG
    void SCDebugBreak( void );
#endif
```

```
class RefCount {
public:
    RefCount() : refcnt_(0)
    {
#ifdef _DEBUG
        magic_ = 0xbabaabab;
#endif
    }
    RefCount(const RefCount &) : refcnt_(0)
    {
#ifdef _DEBUG
        magic_ = 0xbabaabab;
#endif
    }

```

```
    virtual ~RefCount()
    {
        if ( refcnt_ )
            throw( -1 );
    }

```

```
    int decref() // return 1 if it should be deleted
    {
#ifdef _DEBUG
        static void* test = 0;
        if ( this == test )
            SCDebugBreak();
#endif
        return --refcnt_ <= 0;
    }

```

```
    void incref()
    {
#ifdef _DEBUG
        static void* test = 0;
        if ( this == test )
            SCDebugBreak();
#endif
        ++refcnt_;
    }

```

```
    int refcnt()
    {
        return refcnt_;
    }

```

```
#ifdef _DEBUG
    unsigned magic()
    {
        return magic_;
    }
#endif

```

```
private:
#ifdef _DEBUG
    unsigned magic_;
#endif
    int refcnt_;
};
```

```

////////////////////////////////////
// The following are classes to maintain safe reference count on
// classes derived from RefCount

// T must have RefCount as a public base class
// T may be an incomplete type

template<class T>
class ConstRefCountPtr {
public:
    ConstRefCountPtr() : p_(0) { }
    ConstRefCountPtr( T* p ) : p_( p )
    {
        if ( p_ )
            p_>incrcf();
    }

    ~ConstRefCountPtr()
    {
        clear();
    }
    ConstRefCountPtr( const ConstRefCountPtr<T>& p ) :
        p_(p.p_)
    {
        if (p.p_)
            p.p_>incrcf();
    }
    ConstRefCountPtr<T> &operator=( const ConstRefCountPtr<T> & p )
    {
        if ( this != &p ) {
            if (p.p_)
                p.p_>incrcf();

            clear();

            p_ = p.p_;
        }
        return *this;
    }

    void    moveTo( ConstRefCountPtr<T> &dst )
    {
        if ( this == &dst )
            return;

        dst.clear();

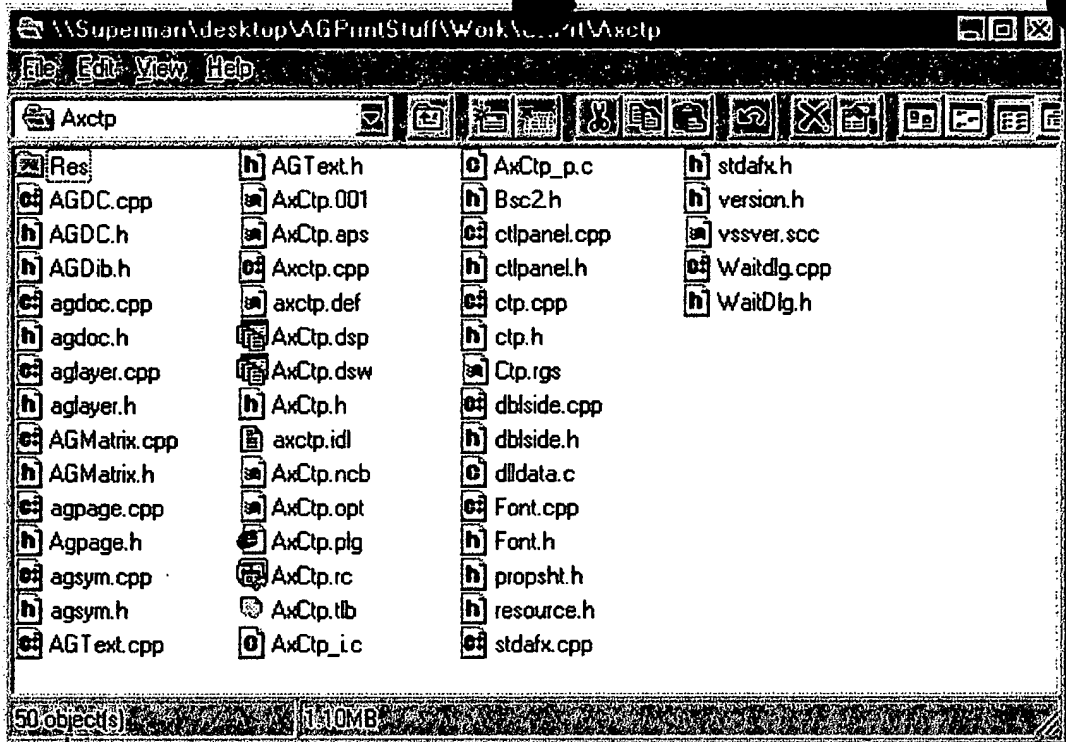
        dst.p_ = p_;
        p_ = 0;
    }

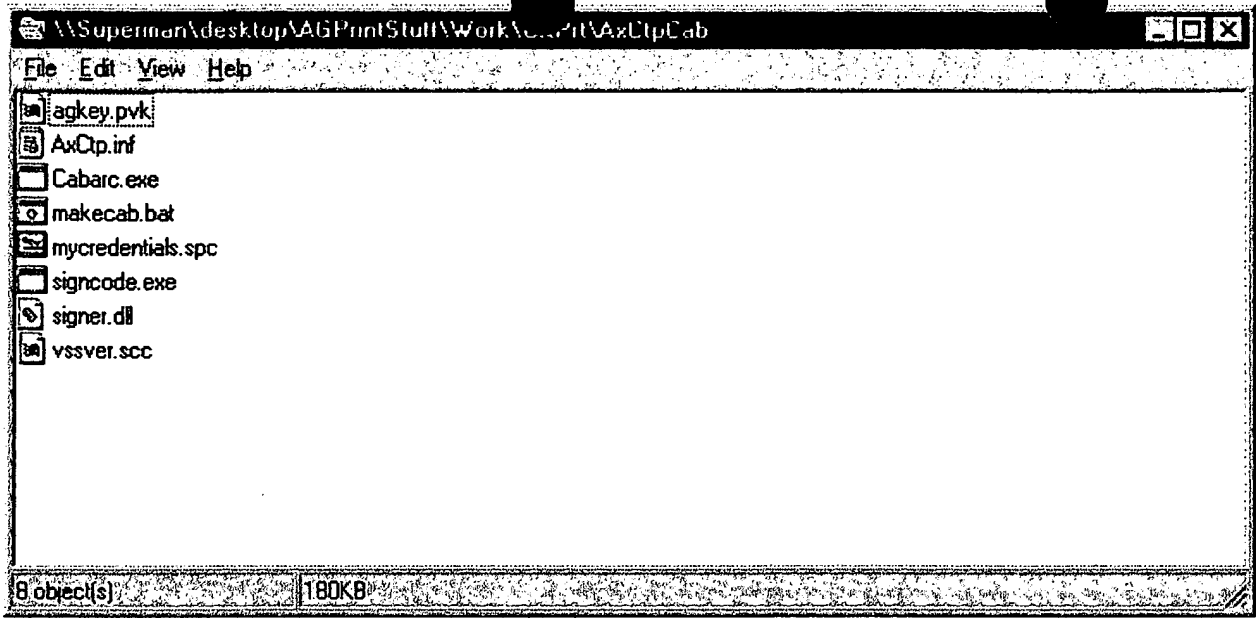
    void    clear()
    {
        if ( p_ && p_>decrcf() )
            delete p_;
        p_ = 0;
    }

    const T* ptr() const
    {
        return p_;
    }
    const T* operator->() const
    {
        return p_;
    }
    const T& operator*() const
    {
        return *p_;
    }
    int isNull() const

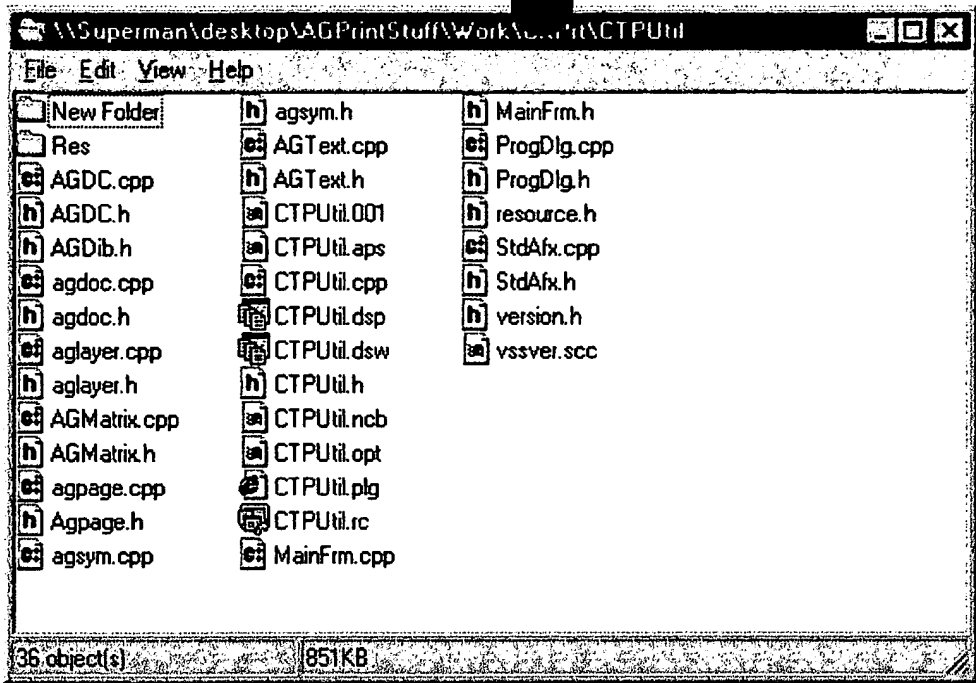
```

$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}} \right) = \frac{\partial L}{\partial x}$

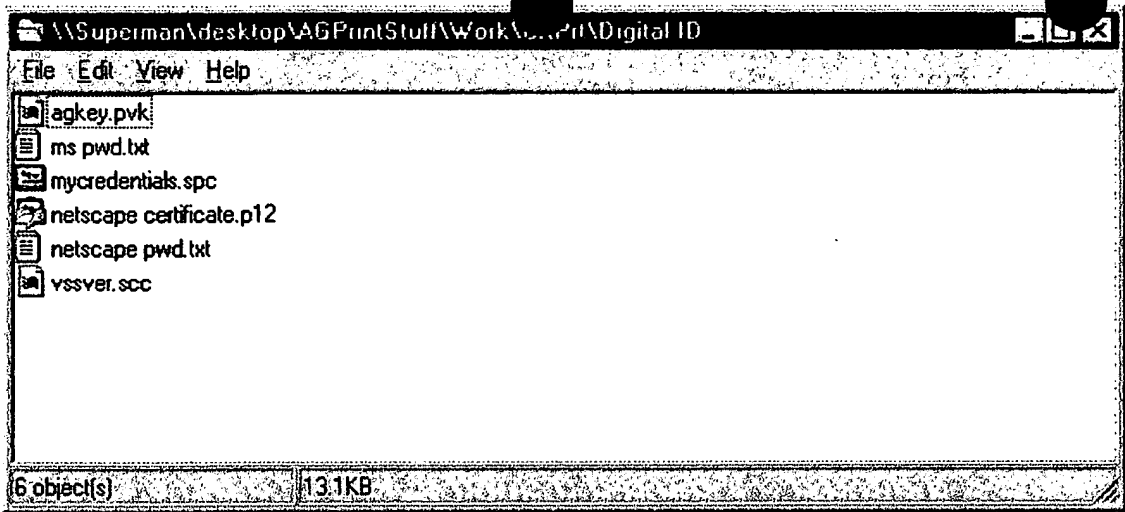




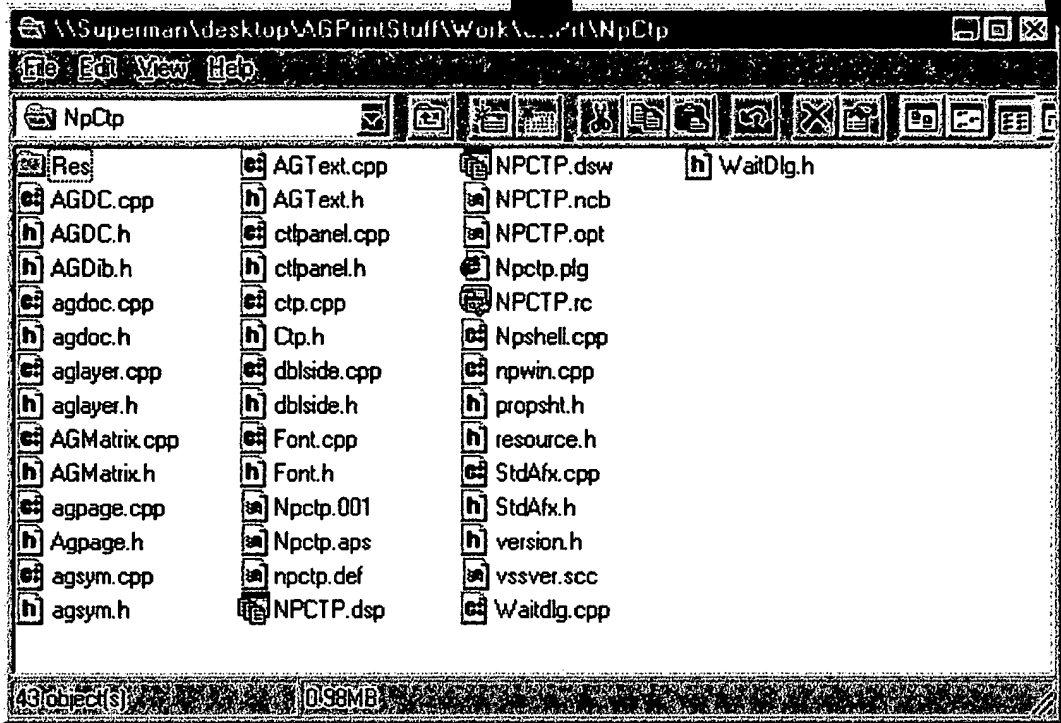
agkey.pvk
AxUtp.inf
Cabarc.exe
makecab.bat
mycredentials.spc
signcode.exe
signer.dll
vssver.scc



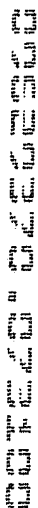
agsym.h
AGText.cpp
AGText.h
CTPUtil.001
CTPUtil.aps
CTPUtil.cpp
CTPUtil.dsp
CTPUtil.dsw
CTPUtil.h
CTPUtil.ncb
CTPUtil.opt
CTPUtil.plg
CTPUtil.rc
MainFrm.cpp
MainFrm.h
ProgDlg.cpp
ProgDlg.h
resource.h
StdAfx.cpp
StdAfx.h
version.h
vssver.scc

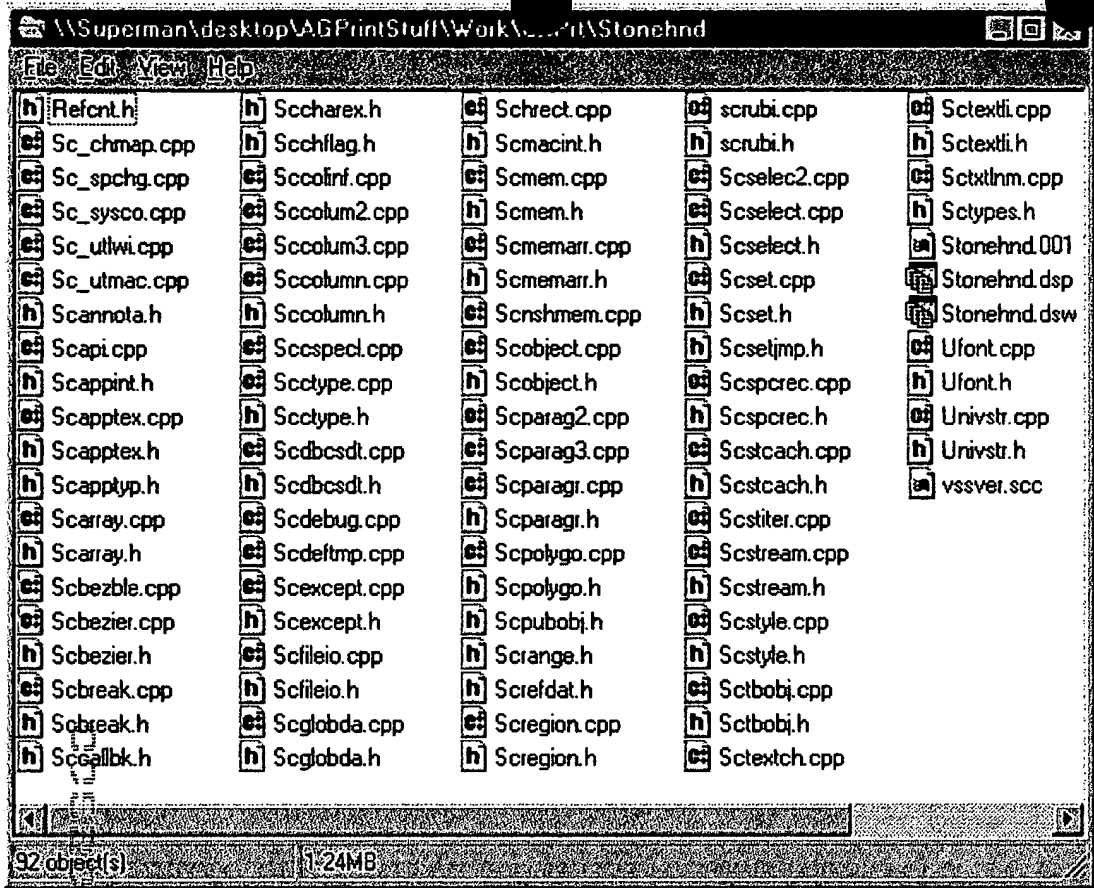


agkey.pvk
ms pwd.txt
mycredentials.spc
netscape certificate.p12
netscape pwd.txt
vssver.scc

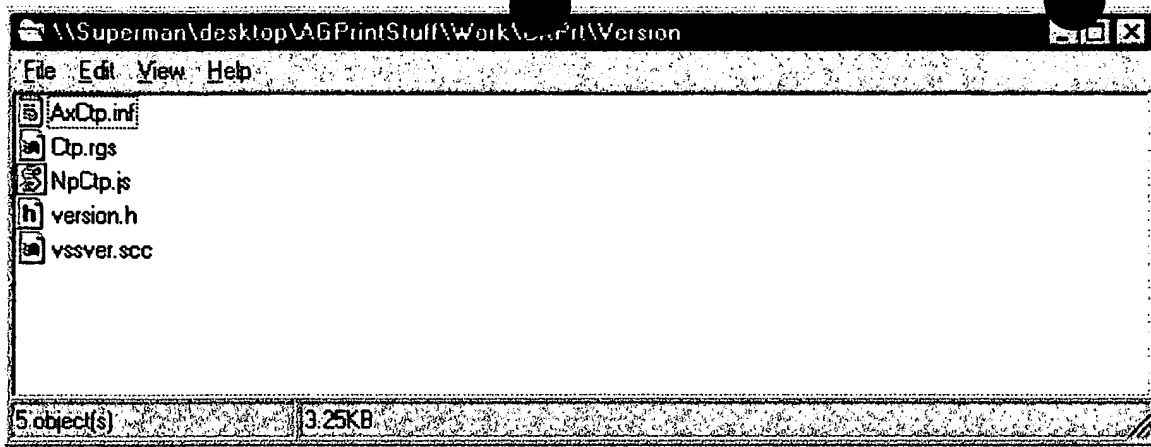


\\Superman\\desktop\\AGPrintStuff\\Work\\NpCtp

[illegible]



Sccharex.h
Scchflag.h
Sccolinf.cpp
Sccolum2.cpp
Sccolum3.cpp
Sccolumn.cpp
Sccolumn.h
Sccspect.cpp
Scctype.cpp
Scctype.h
Scdbcsdt.cpp
Scdbcsdt.h
Scdebug.cpp
Scdefmp.cpp
Scexcept.cpp
Scexcept.h
Scfileio.cpp
Scfileio.h
Scglobda.cpp
Scglobda.h





zcomp.dsw
zcomp.dsp
zcomp.cpp
vssver.scc

\\Superman\\desktop\\A6PrintStuff\\Work\\ZLib

File Edit View Help

adler32.c	inffast.h	zconf.h
compress.c	infixed.h	ZLib.001
crc32.c	inflate.c	ZLib.dsp
deflate.c	inftrees.c	ZLib.dsw
deflate.h	inftrees.h	zlib.h
gzio.c	infutil.c	zutil.c
inffblock.c	infutil.h	zutil.h
inffblock.h	trees.c	
infcodes.c	trees.h	
infcodes.h	uncompr.c	
inffast.c	vssver.scc	

29 object(s)

291KB

\\Superman\\desktop\\A6PrintStuff\\Work\\ZLib

```

/* deflate.h -- internal compression state
 * Copyright (C) 1995-1998 Jean-loup Gailly
 * For conditions of distribution and use, see copyright notice in zlib.h
 */

/* WARNING: this file should *not* be used by applications. It is
   part of the implementation of the compression library and is
   subject to change. Applications should only use zlib.h.
 */

/* @(#) $Id$ */

#ifndef _DEFLATE_H
#define _DEFLATE_H

#include "zutil.h"

/* =====
 * Internal compression state.
 */

#define LENGTH_CODES 29
/* number of length codes, not counting the special END_BLOCK code */

#define LITERALS 256
/* number of literal bytes 0..255 */

#define L_CODES (LITERALS+1+LENGTH_CODES)
/* number of Literal or Length codes, including the END_BLOCK code */

#define D_CODES 30
/* number of distance codes */

#define BL_CODES 19
/* number of codes used to transfer the bit lengths */

#define HEAP_SIZE (2*L_CODES+1)
/* maximum heap size */

#define MAX_BITS 15
/* All codes must not exceed MAX_BITS bits */

#define INIT_STATE 42
#define BUSY_STATE 113
#define FINISH_STATE 666
/* Stream status */

/* Data structure describing a single value and its code string. */
typedef struct ct_data_s {
    union {
        ush freq;       /* frequency count */
        ush code;        /* bit string */
    } fc;
    union {
        ush dad;        /* father node in Huffman tree */
        ush len;         /* length of bit string */
    } dl;
} FAR ct_data;

#define Freq fc.freq
#define Code fc.code
#define Dad dl.dad
#define Len dl.len

typedef struct static_tree_desc_s static_tree_desc;

typedef struct tree_desc_s {
    ct_data *dyn_tree; /* the dynamic tree */
    int max_code;       /* largest code with non zero frequency */
    static_tree_desc *stat_desc; /* the corresponding static tree */
} FAR tree_desc;

```

```
typedef ush Pos;
typedef Pos FAR Posf;
typedef unsigned IPos;
```

```
/* A Pos is an index in the character window. We use short instead of int to
 * save space in the various tables. IPos is used only for parameter passing.
 */
```

```
typedef struct internal_state {
    z_streamp strm;      /* pointer back to this zlib stream */
    int status;          /* as the name implies */
    Bytef *pending_buf;  /* output still pending */
    ulg pending_buf_size; /* size of pending_buf */
    Bytef *pending_out;   /* next pending byte to output to the stream */
    int pending;          /* nb of bytes in the pending buffer */
    int noheader;         /* suppress zlib header and Adler32 */
    Byte data_type;       /* UNKNOWN, BINARY or ASCII */
    Byte method;          /* STORED (for zip only) or DEFLATED */
    int last_flush;       /* value of flush param for previous deflate call */
```

```
    /* used by deflate.c: */
```

```
    uInt w_size;          /* LZ77 window size (32K by default) */
    uInt w_bits;           /* log2(w_size) (8..16) */
    uInt w_mask;           /* w_size - 1 */
```

```
    Bytef *window;
```

```
    /* Sliding window. Input bytes are read into the second half of the window,
     * and move to the first half later to keep a dictionary of at least wSize
     * bytes. With this organization, matches are limited to a distance of
     * wSize-MAX_MATCH bytes, but this ensures that IO is always
     * performed with a length multiple of the block size. Also, it limits
     * the window size to 64K, which is quite useful on MSDOS.
     * To do: use the user input buffer as sliding window.
     */
```

```
    ulg window_size;
```

```
    /* Actual size of window: 2*wSize, except when the user input buffer
     * is directly used as sliding window.
     */
```

```
    Posf *prev;
```

```
    /* Link to older string with same hash index. To limit the size of this
     * array to 64K, this link is maintained only for the last 32K strings.
     * An index in this array is thus a window index modulo 32K.
     */
```

```
    Posf *head; /* Heads of the hash chains or NIL. */
```

```
    uInt ins_h;           /* hash index of string to be inserted */
    uInt hash_size;        /* number of elements in hash table */
    uInt hash_bits;        /* log2(hash_size) */
    uInt hash_mask;        /* hash_size-1 */
```

```
    uInt hash_shift;
```

```
    /* Number of bits by which ins_h must be shifted at each input
     * step. It must be such that after MIN_MATCH steps, the oldest
     * byte no longer takes part in the hash key, that is:
     * hash_shift * MIN_MATCH >= hash_bits
     */
```

```
    long block_start;
```

```
    /* Window position at the beginning of the current output block. Gets
     * negative when the window is moved backwards.
     */
```

```
    uInt match_length;      /* length of best match */
    IPos prev_match;        /* previous match */
    int match_available;    /* set if previous match exists */
    uInt strstart;          /* start of string to insert */
    uInt match_start;       /* start of matching string */
    uInt lookahead;         /* number of valid bytes ahead in window */
```

```

uint prev_length;
/* Length of the best match at previous step. Matches not greater than this
 * are discarded. This is used in the lazy match evaluation.
 */

uint max_chain_length;
/* To speed up deflation, hash chains are never searched beyond this
 * length. A higher limit improves compression ratio but degrades the
 * speed.
 */

uint max_lazy_match;
/* Attempt to find a better match only when the current match is strictly
 * smaller than this value. This mechanism is used only for compression
 * levels >= 4.
 */
# define max_insert_length max_lazy_match
/* Insert new strings in the hash table only if the match length is not
 * greater than this length. This saves time but degrades compression.
 * max_insert_length is used only for compression levels <= 3.
 */

int level; /* compression level (1..9) */
int strategy; /* favor or force Huffman coding*/

uint good_match;
/* Use a faster search when the previous match is longer than this */

int nice_match; /* Stop searching when current match exceeds this */

/* used by trees.c: */
/* Didn't use ct_data typedef below to suppress compiler warning */
struct ct_data_s dyn_ltree[HEAP_SIZE]; /* literal and length tree */
struct ct_data_s dyn_dtree[2*D_CODES+1]; /* distance tree */
struct ct_data_s bl_tree[2*BL_CODES+1]; /* Huffman tree for bit lengths */

struct tree_desc_s l_desc; /* desc. for literal tree */
struct tree_desc_s d_desc; /* desc. for distance tree */
struct tree_desc_s bl_desc; /* desc. for bit length tree */

ush bl_count[MAX_BITS+1];
/* number of codes at each bit length for an optimal tree */

int heap[2*L_CODES+1]; /* heap used to build the Huffman trees */
int heap_len; /* number of elements in the heap */
int heap_max; /* element of largest frequency */
/* The sons of heap[n] are heap[2*n] and heap[2*n+1]. heap[0] is not used.
 * The same heap array is used to build all trees.
 */

uch depth[2*L_CODES+1];
/* Depth of each subtree used as tie breaker for trees of equal frequency
 */

uchf *l_buf; /* buffer for literals or lengths */

uint lit_bufsize;
/* Size of match buffer for literals/lengths. There are 4 reasons for
 * limiting lit_bufsize to 64K:
 * - frequencies can be kept in 16 bit counters
 * - if compression is not successful for the first block, all input
 * data is still in the window so we can still emit a stored block even
 * when input comes from standard input. (This can also be done for
 * all blocks if lit_bufsize is not greater than 32K.)
 * - if compression is not successful for a file smaller than 64K, we can
 * even emit a stored file instead of a stored block (saving 5 bytes).
 * This is applicable only for zip (not gzip or zlib).
 * - creating new Huffman trees less frequently may not provide fast
 * adaptation to changes in the input data statistics. (Take for
 * example a binary file with poorly compressible code followed by
 * a highly compressible string table.) Smaller buffer sizes give
 * fast adaptation but have of course the overhead of transmitting
 * trees more frequently.

```

```

    * - I can't count above
    */

uint last_lit;      /* running index in l_buf */

ushf *d_buf;
/* Buffer for distances. To simplify the code, d_buf and l_buf have
 * the same number of elements. To use different lengths, an extra flag
 * array would be necessary.
 */

ulg opt_len;        /* bit length of current block with optimal trees */
ulg static_len;      /* bit length of current block with static trees */
uint matches;        /* number of string matches in current block */
int last_eob_len;    /* bit length of EOB code for last block */

#ifdef DEBUG
    ulg compressed_len; /* total bit length of compressed file mod 2^32 */
    ulg bits_sent;      /* bit length of compressed data sent mod 2^32 */
#endif

ush bi_buf;
/* Output buffer. bits are inserted starting at the bottom (least
 * significant bits).
 */
int bi_valid;
/* Number of valid bits in bi_buf. All bits above the last valid bit
 * are always zero.
 */

FAR deflate_state;

/* Output a byte on the stream.
 * IN assertion: there is enough room in pending_buf.
 */
#define put_byte(s, c) {s->pending_buf[s->pending++] = (c);}

#define MIN_LOOKAHEAD (MAX_MATCH+MIN_MATCH+1)
/* Minimum amount of lookahead, except at the end of the input file.
 * See deflate.c for comments about the MIN_MATCH+1.
 */

#define MAX_DIST(s) ((s)->w_size-MIN_LOOKAHEAD)
/* In order to simplify the code, particularly on 16 bit machines, match
 * distances are limited to MAX_DIST instead of WSIZE.
 */

/* in trees.c */
void _tr_init      OF((deflate_state *s));
int  _tr_tally     OF((deflate_state *s, unsigned dist, unsigned lc));
void _tr_flush_block OF((deflate_state *s, charf *buf, ulg stored_len,
                        int eof));
void _tr_align     OF((deflate_state *s));
void _tr_stored_block OF((deflate_state *s, charf *buf, ulg stored_len,
                        int eof));

#define d_code(dist) \
    ((dist) < 256 ? _dist_code[dist] : _dist_code[256+((dist)>>7)])
/* Mapping from a distance to a distance code. dist is the distance - 1 and
 * must not have side effects. _dist_code[256] and _dist_code[257] are never
 * used.
 */

#ifdef DEBUG
/* Inline versions of _tr_tally for speed: */

#if defined(GEN_TREES_H) || !defined(STDC)
    extern uch _length_code[];
    extern uch _dist_code[];
#else
    extern const uch _length_code[];
    extern const uch _dist_code[];

```

```
# define _tr_tally_lit(s, c, flush) \
{ uch cc = (c); \
  s->d_buf[s->last_lit] = 0; \
  s->l_buf[s->last_lit++] = cc; \
  s->dyn_ltree[cc].Freq++; \
  flush = (s->last_lit == s->lit_bufsize-1); \
}

# define _tr_tally_dist(s, distance, length, flush) \
{ uch len = (length); \
  uch dist = (distance); \
  s->d_buf[s->last_lit] = dist; \
  s->l_buf[s->last_lit++] = len; \
  dist--; \
  s->dyn_ltree[_length_code[len]+LITERALS+1].Freq++; \
  s->dyn_dtree[d_code(dist)].Freq++; \
  flush = (s->last_lit == s->lit_bufsize-1); \
}

#else
# define _tr_tally_lit(s, c, flush) flush = _tr_tally(s, 0, c)
# define _tr_tally_dist(s, distance, length, flush) \
        flush = _tr_tally(s, distance, length)
#endif

#endif
```

[illegible]

```

/* deflate.c -- compress data using the deflation algorithm
 * Copyright (C) 1995-1998 Jean-loup Gailly.
 * For conditions of distribution and use, see copyright notice in zlib.h
 */

```

```

/*
 * ALGORITHM
 *
 * The "deflation" process depends on being able to identify portions
 * of the input text which are identical to earlier input (within a
 * sliding window trailing behind the input currently being processed).
 *
 * The most straightforward technique turns out to be the fastest for
 * most input files: try all possible matches and select the longest.
 * The key feature of this algorithm is that insertions into the string
 * dictionary are very simple and thus fast, and deletions are avoided
 * completely. Insertions are performed at each input character, whereas
 * string matches are performed only when the previous match ends. So it
 * is preferable to spend more time in matches to allow very fast string
 * insertions and avoid deletions. The matching algorithm for small
 * strings is inspired from that of Rabin & Karp. A brute force approach
 * is used to find longer strings when a small match has been found.
 * A similar algorithm is used in comic (by Jan-Mark Wams) and freeze
 * (by Leonid Broukhis).
 *
 * A previous version of this file used a more sophisticated algorithm
 * (by Fiala and Greene) which is guaranteed to run in linear amortized
 * time, but has a larger average cost, uses more memory and is patented.
 * However the F&G algorithm may be faster for some highly redundant
 * files if the parameter max_chain_length (described below) is too large.

```

ACKNOWLEDGEMENTS

The idea of lazy evaluation of matches is due to Jan-Mark Wams, and I found it in 'freeze' written by Leonid Broukhis. Thanks to many people for bug reports and testing.

REFERENCES

Deutsch, L.P., "DEFLATE Compressed Data Format Specification". Available in ftp://ds.internic.net/rfc/rfc1951.txt

A description of the Rabin and Karp algorithm is given in the book "Algorithms" by R. Sedgewick, Addison-Wesley, p252.

Fiala, E.R., and Greene, D.H. Data Compression with Finite Windows, Comm.ACM, 32,4 (1989) 490-595

```

/* @(#) $Id$ */

```

```

#include "deflate.h"

```

```

const char deflate_copyright[] =
    " deflate 1.1.3 Copyright 1995-1998 Jean-loup Gailly ";

```

```

/*
 * If you use the zlib library in a product, an acknowledgment is welcome
 * in the documentation of your product. If for some reason you cannot
 * include such an acknowledgment, I would appreciate that you keep this
 * copyright string in the executable of your product.
 */

```

```

/* =====
 * Function prototypes.
 */
typedef enum {
    need_more,      /* block not completed, need more input or more output */
    block_done,     /* block flush performed */
    finish_started, /* finish started, need only more output at next deflate */
    finish_done,    /* finish done, accept no more input or output */
} block_state;

typedef block_state (*compress_func) OF((deflate_state *s, int flush));

```

```

/* Compression function. Return the block state after the call. */

local void fill_window      OF((deflate_state *s));
local block_state deflate_stored OF((deflate_state *s, int flush));
local block_state deflate_fast  OF((deflate_state *s, int flush));
local block_state deflate_slow  OF((deflate_state *s, int flush));
local void lm_init          OF((deflate_state *s));
local void putShortMSB      OF((deflate_state *s, uInt b));
local void flush_pending    OF((z_stream strm));
local int read_buf         OF((z_stream strm, Bytef *buf, unsigned size));
#ifdef ASMV
    void match_init OF((void)); /* asm code initialization */
    uInt longest_match OF((deflate_state *s, IPos cur_match));
#else
local uInt longest_match OF((deflate_state *s, IPos cur_match));
#endif

#ifdef DEBUG
local void check_match OF((deflate_state *s, IPos start, IPos match,
                          int length));
#endif

/* =====
 * Local data
 */

#define NIL 0
/* Tail of hash chains */

#ifdef TOO_FAR
    define TOO_FAR 4096
#endif
/* Matches of length 3 are discarded if their distance exceeds TOO_FAR */

#define MIN_LOOKAHEAD (MAX_MATCH+MIN_MATCH+1)
/* Minimum amount of lookahead, except at the end of the input file.
 * See deflate.c for comments about the MIN_MATCH+1.
 */

/* Values for max_lazy_match, good_match and max_chain_length, depending on
 * the desired pack level (0..9). The values given below have been tuned to
 * exclude worst case performance for pathological files. Better values may be
 * found for specific files.
 */

typedef struct config_s {
    uSh good_length; /* reduce lazy search above this match length */
    uSh max_lazy;    /* do not perform lazy search above this match length */
    uSh nice_length; /* quit search above this match length */
    uSh max_chain;
    compress_func func;
} config;

local const config configuration_table[10] = {
    /* good lazy nice chain */
    /* 0 */ {0, 0, 0, 0, deflate_stored}, /* store only */
    /* 1 */ {4, 4, 8, 4, deflate_fast}, /* maximum speed, no lazy matches */
    /* 2 */ {4, 5, 16, 8, deflate_fast},
    /* 3 */ {4, 6, 32, 32, deflate_fast},

    /* 4 */ {4, 4, 16, 16, deflate_slow}, /* lazy matches */
    /* 5 */ {8, 16, 32, 32, deflate_slow},
    /* 6 */ {8, 16, 128, 128, deflate_slow},
    /* 7 */ {8, 32, 128, 256, deflate_slow},
    /* 8 */ {32, 128, 258, 1024, deflate_slow},
    /* 9 */ {32, 258, 258, 4096, deflate_slow}; /* maximum compression */

/* Note: the deflate() code requires max_lazy >= MIN_MATCH and max_chain >= 4
 * For deflate_fast() (levels <= 3) good is ignored and lazy has a different
 * meaning.
 */

#define EQUAL 0
/* result of memcmp for equal strings */

```



```

struct static_tree_desc_s {int dummy;}; /* for buggy compilers */

/* =====
 * Update a hash value with the given input byte
 * IN assertion: all calls to to UPDATE_HASH are made with consecutive
 *   input characters, so that a running hash key can be computed from the
 *   previous key instead of complete recalculation each time.
 */
#define UPDATE_HASH(s,h,c) (h = (((h)<<(s->hash_shift) ^ (c)) & s->hash_mask)

/* =====
 * Insert string str in the dictionary and set match_head to the previous head
 * of the hash chain (the most recent string with same hash key). Return
 * the previous length of the hash chain.
 * If this file is compiled with -DFASTEST, the compression level is forced
 * to 1, and no hash chains are maintained.
 * IN assertion: all calls to to INSERT_STRING are made with consecutive
 *   input characters and the first MIN_MATCH bytes of str are valid
 *   (except for the last MIN_MATCH-1 bytes of the input file).
 */
#ifdef FASTEST
#define INSERT_STRING(s, str, match_head) \
    (UPDATE_HASH(s, s->ins_h, s->window[(str) + (MIN_MATCH-1)]), \
     match_head = s->head[s->ins_h], \
     s->head[s->ins_h] = (Pos)(str))
#else
#define INSERT_STRING(s, str, match_head) \
    (UPDATE_HASH(s, s->ins_h, s->window[(str) + (MIN_MATCH-1)]), \
     s->prev[(str) & s->w_mask] = match_head = s->head[s->ins_h], \
     s->head[s->ins_h] = (Pos)(str))
#endif

/* =====
 * Initialize the hash table (avoiding 64K overflow for 16 bit systems).
 * prev[] will be initialized on the fly.
 */
#define CLEAR_HASH(s) \
    s->head[s->hash_size-1] = NIL; \
    zmemzero((Bytef *)s->head, (unsigned)(s->hash_size-1)*sizeof(*s->head));

/* ===== */
int ZEXPORT deflateInit_(strm, level, version, stream_size)
    z_streamp strm;
    int level;
    const char *version;
    int stream_size;
{
    return deflateInit2_(strm, level, Z_DEFLATED, MAX_WBITS, DEF_MEM_LEVEL,
        Z_DEFAULT_STRATEGY, version, stream_size);
    /* To do: ignore strm->next_in if we use it as window */
}

/* ===== */
int ZEXPORT deflateInit2_(strm, level, method, windowBits, memLevel, strategy,
    version, stream_size)
    z_streamp strm;
    int level;
    int method;
    int windowBits;
    int memLevel;
    int strategy;
    const char *version;
    int stream_size;
{
    deflate_state *s;
    int noheader = 0;
    static const char* my_version = ZLIB_VERSION;

    ushf *overlay;
    /* We overlay pending_buf and d_buf+l_buf. This works since the average
     * output size for (length,distance) codes is <= 24 bits.

```

```

    */

    if (version == Z_NULL || version[0] != my_version[0] ||
        stream_size != sizeof(z_stream)) {
        return Z_VERSION_ERROR;
    }
    if (strm == Z_NULL) return Z_STREAM_ERROR;

    strm->msg = Z_NULL;
    if (strm->zalloc == Z_NULL) {
        strm->zalloc = zcalloc;
        strm->opaque = (voidpf)0;
    }
    if (strm->zfree == Z_NULL) strm->zfree = zcfree;

    if (level == Z_DEFAULT_COMPRESSION) level = 6;
#ifdef FASTEST
    level = 1;
#endif

    if (windowBits < 0) { /* undocumented feature: suppress zlib header */
        noheader = 1;
        windowBits = -windowBits;
    }
    if (memLevel < 1 || memLevel > MAX_MEM_LEVEL || method != Z_DEFLATED ||
        windowBits < 8 || windowBits > 15 || level < 0 || level > 9 ||
        strategy < 0 || strategy > Z_HUFFMAN_ONLY) {
        return Z_STREAM_ERROR;
    }
    s = (deflate_state *) ZALLOC(strm, 1, sizeof(deflate_state));
    if (s == Z_NULL) return Z_MEM_ERROR;
    strm->state = (struct internal_state FAR *)s;
    s->strm = strm;

    s->noheader = noheader;
    s->w_bits = windowBits;
    s->w_size = 1 << s->w_bits;
    s->w_mask = s->w_size - 1;

    s->hash_bits = memLevel + 7;
    s->hash_size = 1 << s->hash_bits;
    s->hash_mask = s->hash_size - 1;
    s->hash_shift = ((s->hash_bits+MIN_MATCH-1)/MIN_MATCH);

    s->window = (Bytef *) ZALLOC(strm, s->w_size, 2*sizeof(Byte));
    s->prev = (Posf *) ZALLOC(strm, s->w_size, sizeof(Pos));
    s->head = (Posf *) ZALLOC(strm, s->hash_size, sizeof(Pos));

    s->lit_bufsize = 1 << (memLevel + 6); /* 16K elements by default */

    overlay = (ushf *) ZALLOC(strm, s->lit_bufsize, sizeof(ush)+2);
    s->pending_buf = (uchf *) overlay;
    s->pending_buf_size = (ulg)s->lit_bufsize * (sizeof(ush)+2L);

    if (s->window == Z_NULL || s->prev == Z_NULL || s->head == Z_NULL ||
        s->pending_buf == Z_NULL) {
        strm->msg = (char*)ERR_MSG(Z_MEM_ERROR);
        deflateEnd (strm);
        return Z_MEM_ERROR;
    }
    s->d_buf = overlay + s->lit_bufsize/sizeof(ush);
    s->l_buf = s->pending_buf + (1+sizeof(ush))*s->lit_bufsize;

    s->level = level;
    s->strategy = strategy;
    s->method = (Byte)method;

    return deflateReset(strm);
}

/* ===== */
int ZEXPORT deflateSetDictionary (strm, dictionary, dictLength)
    z_streamp strm;

```

```

const Bytef *dictionary;
uint dictLength;

{
    deflate_state *s;
    uint length = dictLength;
    uint n;
    IPos hash_head = 0;

    if (strm == Z_NULL || strm->state == Z_NULL || dictionary == Z_NULL ||
        strm->state->status != INIT_STATE) return Z_STREAM_ERROR;

    s = strm->state;
    strm->adler = Adler32(strm->adler, dictionary, dictLength);

    if (length < MIN_MATCH) return Z_OK;
    if (length > MAX_DIST(s)) {
        length = MAX_DIST(s);
#ifdef USE_DICT_HEAD
        dictionary += dictLength - length; /* use the tail of the dictionary */
#endif
    }
    zmemcpy(s->window, dictionary, length);
    s->strstart = length;
    s->block_start = (long)length;

    /* Insert all strings in the hash table (except for the last two bytes).
     * s->lookahead stays null, so s->ins_h will be recomputed at the next
     * call of fill_window.
     */
    s->ins_h = s->window[0];
    UPDATE_HASH(s, s->ins_h, s->window[1]);
    for (n = 0; n <= length - MIN_MATCH; n++) {
        INSERT_STRING(s, n, hash_head);
    }
    if (hash_head) hash_head = 0; /* to make compiler happy */
    return Z_OK;
}

===== */
int ZEXPORT deflateReset (strm)
    z_stream strm;
{
    deflate_state *s;

    if (strm == Z_NULL || strm->state == Z_NULL ||
        strm->zalloc == Z_NULL || strm->zfree == Z_NULL) return Z_STREAM_ERROR;

    strm->total_in = strm->total_out = 0;
    strm->msg = Z_NULL; /* use zfree if we ever allocate msg dynamically */
    strm->data_type = Z_UNKNOWN;

    s = (deflate_state *)strm->state;
    s->pending = 0;
    s->pending_out = s->pending_buf;

    if (s->noheader < 0) {
        s->noheader = 0; /* was set to -1 by deflate(..., Z_FINISH); */
    }
    s->status = s->noheader ? BUSY_STATE : INIT_STATE;
    strm->adler = 1;
    s->last_flush = Z_NO_FLUSH;

    _tr_init(s);
    lm_init(s);

    return Z_OK;
}

===== */
int ZEXPORT deflateParams(strm, level, strategy)
    z_stream strm;
    int level;
    int strategy;

```

```

{
    deflate_state *s;
    compress_func func;
    int err = Z_OK;

    if (strm == Z_NULL || strm->state == Z_NULL) return Z_STREAM_ERROR;
    s = strm->state;

    if (level == Z_DEFAULT_COMPRESSION) {
        level = 6;
    }
    if (level < 0 || level > 9 || strategy < 0 || strategy > Z_HUFFMAN_ONLY) {
        return Z_STREAM_ERROR;
    }
    func = configuration_table[s->level].func;

    if (func != configuration_table[level].func && strm->total_in != 0) {
        /* Flush the last buffer: */
        err = deflate(strm, Z_PARTIAL_FLUSH);
    }
    if (s->level != level) {
        s->level = level;
        s->max_lazy_match = configuration_table[level].max_lazy;
        s->good_match = configuration_table[level].good_length;
        s->nice_match = configuration_table[level].nice_length;
        s->max_chain_length = configuration_table[level].max_chain;
    }
    s->strategy = strategy;
    return err;
}

/* =====
 * Put a short in the pending buffer. The 16-bit value is put in MSB order.
 * IN assertion: the stream state is correct and there is enough room in
 * pending_buf.
 */
local void putShortMSB (s, b)
    deflate_state *s;
    uInt b;
{
    put_byte(s, (Byte)(b >> 8));
    put_byte(s, (Byte)(b & 0xff));
}

/* =====
 * Flush as much pending output as possible. All deflate() output goes
 * through this function so some applications may wish to modify it
 * to avoid allocating a large strm->next_out buffer and copying into it.
 * (See also read_buf()).
 */
local void flush_pending(strm)
    z_streamp strm;
{
    unsigned len = strm->state->pending;

    if (len > strm->avail_out) len = strm->avail_out;
    if (len == 0) return;

    zmemcpy(strm->next_out, strm->state->pending_out, len);
    strm->next_out += len;
    strm->state->pending_out += len;
    strm->total_out += len;
    strm->avail_out -= len;
    strm->state->pending -= len;
    if (strm->state->pending == 0) {
        strm->state->pending_out = strm->state->pending_buf;
    }
}

/* ===== */
int ZEXPORT deflate (strm, flush)
    z_streamp strm;
    int flush;

```

```

{
    int old_flush; /* value of flush param for previous deflate call */
    deflate_state *s;

    if (strm == Z_NULL || strm->state == Z_NULL ||
        flush > Z_FINISH || flush < 0) {
        return Z_STREAM_ERROR;
    }
    s = strm->state;

    if (strm->next_out == Z_NULL ||
        (strm->next_in == Z_NULL && strm->avail_in != 0) ||
        (s->status == FINISH_STATE && flush != Z_FINISH)) {
        ERR_RETURN(strm, Z_STREAM_ERROR);
    }
    if (strm->avail_out == 0) ERR_RETURN(strm, Z_BUF_ERROR);

    s->strm = strm; /* just in case */
    old_flush = s->last_flush;
    s->last_flush = flush;

    /* Write the zlib header */
    if (s->status == INIT_STATE) {

        uInt header = (Z_DEFLATED + ((s->w_bits-8)<<4)) << 8;
        uInt level_flags = (s->level-1) >> 1;

        if (level_flags > 3) level_flags = 3;
        header |= (level_flags << 6);
        if (s->strstart != 0) header |= PRESET_DICT;
        header += 31 - (header % 31);

        s->status = BUSY_STATE;
        putShortMSB(s, header);

        /* Save the Adler32 of the preset dictionary: */
        if (s->strstart != 0) {
            putShortMSB(s, (uInt)(strm->adler >> 16));
            putShortMSB(s, (uInt)(strm->adler & 0xffff));
        }
        strm->adler = 1L;
    }

    /* Flush as much pending output as possible */
    if (s->pending != 0) {
        flush_pending(strm);
        if (strm->avail_out == 0) {
            /* Since avail_out is 0, deflate will be called again with
             * more output space, but possibly with both pending and
             * avail_in equal to zero. There won't be anything to do,
             * but this is not an error situation so make sure we
             * return OK instead of BUF_ERROR at next call of deflate:
             */
            s->last_flush = -1;
            return Z_OK;
        }
    }

    /* Make sure there is something to do and avoid duplicate consecutive
     * flushes. For repeated and useless calls with Z_FINISH, we keep
     * returning Z_STREAM_END instead of Z_BUF_ERROR.
     */
    } else if (strm->avail_in == 0 && flush <= old_flush &&
        flush != Z_FINISH) {
        ERR_RETURN(strm, Z_BUF_ERROR);
    }

    /* User must not provide more input after the first FINISH: */
    if (s->status == FINISH_STATE && strm->avail_in != 0) {
        ERR_RETURN(strm, Z_BUF_ERROR);
    }

    /* Start a new block or continue the current one.
     */

```

```

if (strm->avail_in != 0 || lookahead != 0 ||
    (flush != Z_NO_FLUSH && s->status != FINISH_STATE)) {
    block_state bstate;

    bstate = (*(configuration_table[s->level].func))(s, flush);

    if (bstate == finish_started || bstate == finish_done) {
        s->status = FINISH_STATE;
    }
    if (bstate == need_more || bstate == finish_started) {
        if (strm->avail_out == 0) {
            s->last_flush = -1; /* avoid BUF_ERROR next call, see above */
        }
        return Z_OK;
        /* If flush != Z_NO_FLUSH && avail_out == 0, the next call
        * of deflate should use the same flush parameter to make sure
        * that the flush is complete. So we don't have to output an
        * empty block here, this will be done at next call. This also
        * ensures that for a very small output buffer, we emit at most
        * one empty block.
        */
    }
    if (bstate == block_done) {
        if (flush == Z_PARTIAL_FLUSH) {
            _tr_align(s);
        } else { /* FULL_FLUSH or SYNC_FLUSH */
            _tr_stored_block(s, (char*)0, 0L, 0);
            /* For a full flush, this empty block will be recognized
            * as a special marker by inflate_sync().
            */
            if (flush == Z_FULL_FLUSH) {
                CLEAR_HASH(s); /* forget history */
            }
        }
        flush_pending(strm);
        if (strm->avail_out == 0) {
            s->last_flush = -1; /* avoid BUF_ERROR at next call, see above */
            return Z_OK;
        }
    }
}
Assert(strm->avail_out > 0, "bug2");

if (flush != Z_FINISH) return Z_OK;
if (s->noheader) return Z_STREAM_END;

/* Write the zlib trailer (adler32) */
putShortMSB(s, (uint)(strm->adler >> 16));
putShortMSB(s, (uint)(strm->adler & 0xffff));
flush_pending(strm);
/* If avail_out is zero, the application will call deflate again
 * to flush the rest.
 */
s->noheader = -1; /* write the trailer only once! */
return s->pending != 0 ? Z_OK : Z_STREAM_END;
}

/* ===== */
int ZEXPORT deflateEnd (strm)
    z_streamp strm;
{
    int status;

    if (strm == Z_NULL || strm->state == Z_NULL) return Z_STREAM_ERROR;

    status = strm->state->status;
    if (status != INIT_STATE && status != BUSY_STATE &&
        status != FINISH_STATE) {
        return Z_STREAM_ERROR;
    }

    /* Deallocate in reverse order of allocations: */
    TRY_FREE(strm, strm->state->pending_buf);

```

```

    TRY_FREE(strm, strm->state->ad);
    TRY_FREE(strm, strm->state->prev);
    TRY_FREE(strm, strm->state->window);

    ZFREE(strm, strm->state);
    strm->state = Z_NULL;

    return status == BUSY_STATE ? Z_DATA_ERROR : Z_OK;
}

/* =====
 * Copy the source state to the destination state.
 * To simplify the source, this is not supported for 16-bit MSDOS (which
 * doesn't have enough memory anyway to duplicate compression states).
 */
int ZEXPORT deflateCopy (dest, source)
    z_stream dest;
    z_stream source;
{
    #ifdef MAXSEG_64K
        return Z_STREAM_ERROR;
    #else
        deflate_state *ds;
        deflate_state *ss;
        ushf *overlay;

        if (source == Z_NULL || dest == Z_NULL || source->state == Z_NULL) {
            return Z_STREAM_ERROR;
        }

        ss = source->state;

        *dest = *source;

        ds = (deflate_state *) ZALLOC(dest, 1, sizeof(deflate_state));
        if (ds == Z_NULL) return Z_MEM_ERROR;
        dest->state = (struct internal_state FAR *) ds;
        *ds = *ss;
        ds->strm = dest;

        ds->window = (Bytef *) ZALLOC(dest, ds->w_size, 2*sizeof(Byte));
        ds->prev = (Posf *) ZALLOC(dest, ds->w_size, sizeof(Pos));
        ds->head = (Posf *) ZALLOC(dest, ds->hash_size, sizeof(Pos));
        overlay = (ushf *) ZALLOC(dest, ds->lit_bufsize, sizeof(ush)+2);
        ds->pending_buf = (uchf *) overlay;

        if (ds->window == Z_NULL || ds->prev == Z_NULL || ds->head == Z_NULL ||
            ds->pending_buf == Z_NULL) {
            deflateEnd (dest);
            return Z_MEM_ERROR;
        }

        /* following memcpy do not work for 16-bit MSDOS */
        memcpy(ds->window, ss->window, ds->w_size * 2 * sizeof(Byte));
        memcpy(ds->prev, ss->prev, ds->w_size * sizeof(Pos));
        memcpy(ds->head, ss->head, ds->hash_size * sizeof(Pos));
        memcpy(ds->pending_buf, ss->pending_buf, (uInt)ds->pending_buf_size);

        ds->pending_out = ds->pending_buf + (ss->pending_out - ss->pending_buf);
        ds->d_buf = overlay + ds->lit_bufsize/sizeof(ush);
        ds->l_buf = ds->pending_buf + (1+sizeof(ush))*ds->lit_bufsize;

        ds->l_desc.dyn_tree = ds->dyn_ltree;
        ds->d_desc.dyn_tree = ds->dyn_dtree;
        ds->bl_desc.dyn_tree = ds->bl_tree;

        return Z_OK;
    #endif
}

/* =====
 * Read a new buffer from the current input stream, update the Adler32
 * and total number of bytes read. All deflate() input goes through

```

```

* this function so some applications may wish to modify it to avoid
* allocating a large strm->next_in buffer and copying from it.
* (See also flush_pending()).
*/

```

```

local int read_buf(strm, buf, size)
    z_streamp strm;
    Bytef *buf;
    unsigned size;
{
    unsigned len = strm->avail_in;

    if (len > size) len = size;
    if (len == 0) return 0;

    strm->avail_in -= len;

    if (!strm->state->noheader) {
        strm->adler = Adler32(strm->adler, strm->next_in, len);
    }
    memcpy(buf, strm->next_in, len);
    strm->next_in += len;
    strm->total_in += len;

    return (int)len;
}

```

```

/* =====
* Initialize the "longest match" routines for a new zlib stream
*/

```

```

local void lm_init (s)
    deflate_state *s;
{
    s->window_size = (ulg)2L*s->w_size;

    CLEAR_HASH(s);

    /* Set the default configuration parameters:
    */
    s->max_lazy_match = configuration_table[s->level].max_lazy;
    s->good_match = configuration_table[s->level].good_length;
    s->nice_match = configuration_table[s->level].nice_length;
    s->max_chain_length = configuration_table[s->level].max_chain;

    s->strstart = 0;
    s->block_start = 0L;
    s->lookahead = 0;
    s->match_length = s->prev_length = MIN_MATCH-1;
    s->match_available = 0;
    s->ins_h = 0;

#ifdef ASMV
    match_init(); /* initialize the asm code */
#endif
}

```

```

/* =====
* Set match_start to the longest match starting at the given string and
* return its length. Matches shorter or equal to prev_length are discarded,
* in which case the result is equal to prev_length and match_start is
* garbage.
* IN assertions: cur_match is the head of the hash chain for the current
* string (strstart) and its distance is <= MAX_DIST, and prev_length >= 1
* OUT assertion: the match length is not greater than s->lookahead.
*/

```

```

#ifdef ASMV
/* For 80x86 and 680x0, an optimized version will be provided in match.asm or
* match.S. The code will be functionally equivalent.
*/

```

```

#ifdef FASTEST
local uint longest_match(s, cur_match)
    deflate_state *s;
    IPos cur_match;
/* current match */
{
    unsigned chain_length = s->max_chain_length; /* max hash chain length */

```



```

register Bytef *scan = s->window + s->strstart; /* current string */
register Bytef *match; /* matched string */
register int len; /* length of current match */
int best_len = s->prev_length; /* best match length so far */
int nice_match = s->nice_match; /* stop if match long enough */
IPos limit = s->strstart > (IPos)MAX_DIST(s) ?
    s->strstart - (IPos)MAX_DIST(s) : NIL;
/* Stop when cur_match becomes <= limit. To simplify the code,
 * we prevent matches with the string of window index 0.
 */
Posf *prev = s->prev;
uInt wmask = s->w_mask;

#ifdef UNALIGNED_OK
/* Compare two bytes at a time. Note: this is not always beneficial.
 * Try with and without -DUNALIGNED_OK to check.
 */
register Bytef *strend = s->window + s->strstart + MAX_MATCH - 1;
register ush scan_start = *(ushf*)scan;
register ush scan_end = *(ushf*)(scan+best_len-1);
#else
register Bytef *strend = s->window + s->strstart + MAX_MATCH;
register Byte scan_end1 = scan[best_len-1];
register Byte scan_end = scan[best_len];
#endif

/* The code is optimized for HASH_BITS >= 8 and MAX_MATCH-2 multiple of 16.
 * It is easy to get rid of this optimization if necessary.
 */
Assert(s->hash_bits >= 8 && MAX_MATCH == 258, "Code too clever");

/* Do not waste too much time if we already have a good match: */
if (s->prev_length >= s->good_match) {
    chain_length >>= 2;
}
/* Do not look for matches beyond the end of the input. This is necessary
 * to make deflate deterministic.
 */
if ((uInt)nice_match > s->lookahead) nice_match = s->lookahead;

Assert((ulg)s->strstart <= s->window_size-MIN_LOOKAHEAD, "need lookahead");

do {
    Assert(cur_match < s->strstart, "no future");
    match = s->window + cur_match;

    /* Skip to next match if the match length cannot increase
     * or if the match length is less than 2:
     */
    if (defined(UNALIGNED_OK) && MAX_MATCH == 258)
        /* This code assumes sizeof(unsigned short) == 2. Do not use
         * UNALIGNED_OK if your compiler uses a different size.
         */
        if (*(ushf*)(match+best_len-1) != scan_end ||
            *(ushf*)match != scan_start) continue;

    /* It is not necessary to compare scan[2] and match[2] since they are
     * always equal when the other bytes match, given that the hash keys
     * are equal and that HASH_BITS >= 8. Compare 2 bytes at a time at
     * strstart+3, +5, ... up to strstart+257. We check for insufficient
     * lookahead only every 4th comparison; the 128th check will be made
     * at strstart+257. If MAX_MATCH-2 is not a multiple of 8, it is
     * necessary to put more guard bytes at the end of the window, or
     * to check more often for insufficient lookahead.
     */
    Assert(scan[2] == match[2], "scan[2]?");
    scan++, match++;
    do {
        } while (*(ushf*)(scan+=2) == *(ushf*)(match+=2) &&
            *(ushf*)(scan+=2) == *(ushf*)(match+=2) &&
            *(ushf*)(scan+=2) == *(ushf*)(match+=2) &&
            *(ushf*)(scan+=2) == *(ushf*)(match+=2) &&
            scan < strend);

```

```

/* The funny "do {}" generates better code on most compilers

/* Here, scan <= window+strstart+257 */
Assert(scan <= s->window+(unsigned)(s->window_size-1), "wild scan");
if (*scan == *match) scan++;

len = (MAX_MATCH - 1) - (int)(strend-scan);
scan = strend - (MAX_MATCH-1);

#else /* UNALIGNED_OK */

if (match[best_len] != scan_end ||
    match[best_len-1] != scan_end1 ||
    *match != *scan ||
    *++match != scan[1]) continue;

/* The check at best_len-1 can be removed because it will be made
 * again later. (This heuristic is not always a win.)
 * It is not necessary to compare scan[2] and match[2] since they
 * are always equal when the other bytes match, given that
 * the hash keys are equal and that HASH_BITS >= 8.
 */
scan += 2, match++;
Assert(*scan == *match, "match[2]?");

/* We check for insufficient lookahead only every 8th comparison;
 * the 256th check will be made at strstart+256.
 */
do {
} while ((*++scan == *++match && *++scan == *++match &&
    *++scan == *++match && *++scan == *++match &&
    *++scan == *++match && *++scan == *++match &&
    *++scan == *++match && *++scan == *++match &&
    scan < strend);

Assert(scan <= s->window+(unsigned)(s->window_size-1), "wild scan");

len = MAX_MATCH - (int)(strend - scan);
scan = strend - MAX_MATCH;

#endif /* UNALIGNED_OK */

if (len > best_len) {
    s->match_start = cur_match;
    best_len = len;
    if (len >= nice_match) break;
#ifdef UNALIGNED_OK
    scan_end = *(ushf*)(scan+best_len-1);
#else
    scan_end1 = scan[best_len-1];
    scan_end = scan[best_len];
#endif
}
} while ((cur_match = prev[cur_match & wmask]) > limit
    && --chain_length != 0);

if ((uInt)best_len <= s->lookahead) return (uInt)best_len;
return s->lookahead;
}

#else /* FASTEST */
/* -----
 * Optimized version for level == 1 only
 */
local uInt longest_match(s, cur_match)
    deflate_state *s;
    IPos cur_match;
    /* current match */
{
    register Bytef *scan = s->window + s->strstart; /* current string */
    register Bytef *match; /* matched string */
    register int len; /* length of current match */
    register Bytef *strend = s->window + s->strstart + MAX_MATCH;

```

```

/* The code is optimized for HASH_BITS >= 8 and MAX_MATCH-2 multiple of 16.
 * It is easy to get rid of this optimization if necessary.
 */
Assert(s->hash_bits >= 8 && MAX_MATCH == 258, "Code too clever");

Assert((ulg)s->strstart <= s->window_size-MIN_LOOKAHEAD, "need lookahead");

Assert(cur_match < s->strstart, "no future");

match = s->window + cur_match;

/* Return failure if the match length is less than 2:
 */
if (match[0] != scan[0] || match[1] != scan[1]) return MIN_MATCH-1;

/* The check at best_len-1 can be removed because it will be made
 * again later. (This heuristic is not always a win.)
 * It is not necessary to compare scan[2] and match[2] since they
 * are always equal when the other bytes match, given that
 * the hash keys are equal and that HASH_BITS >= 8.
 */
scan += 2, match += 2;
Assert(*scan == *match, "match[2]?");

/* We check for insufficient lookahead only every 8th comparison;
 * the 256th check will be made at strstart+258.
 */
do {
} while (++scan == ++match && ++scan == ++match &&
        ++scan == ++match && ++scan == ++match &&
        ++scan == ++match && ++scan == ++match &&
        ++scan == ++match && ++scan == ++match &&
        scan < strend);

Assert(scan <= s->window+(unsigned)(s->window_size-1), "wild scan");

len = MAX_MATCH - (int)(strend - scan);

if (len < MIN_MATCH) return MIN_MATCH - 1;

s->match_start = cur_match;
return len <= s->lookahead ? len : s->lookahead;
#endif /* FASTEST */
#endif /* ASMV */

#ifdef DEBUG
/* -----
 * Check that the match at match_start is indeed a match.
 */
local void check_match(s, start, match, length)
    deflate_state *s;
    IPos start, match;
    int length;
{
    /* check that the match is indeed a match */
    if (memcmp(s->window + match,
               s->window + start, length) != EQUAL) {
        fprintf(stderr, " start %u, match %u, length %d\n",
            start, match, length);
        do {
            fprintf(stderr, "%c%c", s->window[match++], s->window[start++]);
        } while (--length != 0);
        z_error("invalid match");
    }
    if (z_verbose > 1) {
        fprintf(stderr, "\\[%d,%d]", start-match, length);
        do { putc(s->window[start++], stderr); } while (--length != 0);
    }
}
#else
#define check_match(s, start, match, length)
#endif

```

```

/* =====
 * Fill the window when the lookahead becomes insufficient.
 * Updates strstart and lookahead.
 *
 * IN assertion: lookahead < MIN_LOOKAHEAD
 * OUT assertions: strstart <= window_size-MIN_LOOKAHEAD
 *   At least one byte has been read, or avail_in == 0; reads are
 *   performed for at least two bytes (required for the zip translate_eol
 *   option -- not supported here).
 */
local void fill_window(s)
    deflate_state *s;
{
    register unsigned n, m;
    register Posf *p;
    unsigned more; /* Amount of free space at the end of the window. */
    uint wsize = s->w_size;

    do {
        more = (unsigned)(s->window_size - (ulg)s->lookahead - (ulg)s->strstart);

        /* Deal with !@#$% 64K limit: */
        if (more == 0 && s->strstart == 0 && s->lookahead == 0) {
            more = wsize;

        } else if (more == (unsigned)(-1)) {
            /* Very unlikely, but possible on 16 bit machine if strstart == 0
             * and lookahead == 1 (input done one byte at time)
             */
            more--;

        /* If the window is almost full and there is insufficient lookahead,
         * move the upper half to the lower one to make room in the upper half.
         */
        } else if (s->strstart >= wsize+MAX_DIST(s)) {

            zmemcpy(s->window, s->window+wsize, (unsigned)wsize);
            s->match_start -= wsize;
            s->strstart -= wsize; /* we now have strstart >= MAX_DIST */
            s->block_start -= (long) wsize;

            /* Slide the hash table (could be avoided with 32 bit values
             * at the expense of memory usage). We slide even when level == 0
             * to keep the hash table consistent if we switch back to level > 0
             * later. (Using level 0 permanently is not an optimal usage of
             * zlib, so we don't care about this pathological case.)
             */
            n = s->hash_size;
            p = &s->head[n];
            do {
                m = *--p;
                *p = (Pos)(m >= wsize ? m-wsize : NIL);
            } while (--n);

            n = wsize;
            #ifndef FASTEST
            p = &s->prev[n];
            do {
                m = *--p;
                *p = (Pos)(m >= wsize ? m-wsize : NIL);
            /* If n is not on any hash chain, prev[n] is garbage but
             * its value will never be used.
             */
            } while (--n);
            #endif
            more += wsize;
        }
        if (s->strm->avail_in == 0) return;

        /* If there was no sliding:
         *   strstart <= WSIZE+MAX_DIST-1 && lookahead <= MIN_LOOKAHEAD - 1 &&
         *   more == window_size - lookahead - strstart

```

```

    * => more >= window_size - (MIN_LOOKAHEAD-1 + WSIZE + MAX_MATCH-1)
    * => more >= window_size - 2*WSIZE + 2
    * In the BIG_MEM or MMAP case (not yet supported),
    *   window_size == input_size + MIN_LOOKAHEAD  &&
    *   strstart + s->lookahead <= input_size => more >= MIN_LOOKAHEAD.
    * Otherwise, window_size == 2*WSIZE so more >= 2.
    * If there was sliding, more >= WSIZE. So in all cases, more >= 2.
    */
    Assert(more >= 2, "more < 2");

    n = read_buf(s->strm, s->window + s->strstart + s->lookahead, more);
    s->lookahead += n;

    /* Initialize the hash value now that we have some input: */
    if (s->lookahead >= MIN_MATCH) {
        s->ins_h = s->window[s->strstart];
        UPDATE_HASH(s, s->ins_h, s->window[s->strstart+1]);
    }
    #if MIN_MATCH != 3
        Call UPDATE_HASH() MIN_MATCH-3 more times
    #endif
    }
    /* If the whole input has less than MIN_MATCH bytes, ins_h is garbage,
     * but this is not important since only literal bytes will be emitted.
     */

    } while (s->lookahead < MIN_LOOKAHEAD && s->strm->avail_in != 0);
}

/* =====
 * Flush the current block, with given end-of-file flag.
 * IN assertion: strstart is set to the end of the current match.
 */
#define FLUSH_BLOCK_ONLY(s, eof) { \
    _tr_flush_block(s, (s->block_start >= 0L ? \
        (charf *)&s->window[(unsigned)s->block_start] : \
        (charf *)Z_NULL), \
        (ulg)((long)s->strstart - s->block_start), \
        (eof)); \
    s->block_start = s->strstart; \
    flush_pending(s->strm); \
    Tracev((stderr, "[FLUSH]")); \
}

/* Same but force premature exit if necessary. */
#define FLUSH_BLOCK(s, eof) { \
    FLUSH_BLOCK_ONLY(s, eof); \
    if (s->strm->avail_out == 0) return (eof) ? finish_started : need_more; \
}

/* =====
 * Copy without compression as much as possible from the input stream, return
 * the current block state.
 * This function does not insert new strings in the dictionary since
 * uncompressible data is probably not useful. This function is used
 * only for the level=0 compression option.
 * NOTE: this function should be optimized to avoid extra copying from
 * window to pending_buf.
 */
local block_state deflate_stored(s, flush)
    deflate_state *s;
    int flush;
{
    /* Stored blocks are limited to 0xffff bytes, pending_buf is limited
     * to pending_buf_size, and each stored block has a 5 byte header:
     */
    ulg max_block_size = 0xffff;
    ulg max_start;

    if (max_block_size > s->pending_buf_size - 5) {
        max_block_size = s->pending_buf_size - 5;
    }

    /* Copy as much as possible from input to output: */

```

```

for (;;) {
    /* Fill the window as much as possible: */
    if (s->lookahead <= 1) {

        Assert(s->strstart < s->w_size+MAX_DIST(s) ||
            s->block_start >= (long)s->w_size, "slide too late");

        fill_window(s);
        if (s->lookahead == 0 && flush == Z_NO_FLUSH) return need_more;

        if (s->lookahead == 0) break; /* flush the current block */
    }
    Assert(s->block_start >= 0L, "block gone");

    s->strstart += s->lookahead;
    s->lookahead = 0;

    /* Emit a stored block if pending_buf will be full: */
    max_start = s->block_start + max_block_size;
    if (s->strstart == 0 || (ulg)s->strstart >= max_start) {
        /* strstart == 0 is possible when wraparound on 16-bit machine */
        s->lookahead = (uInt)(s->strstart - max_start);
        s->strstart = (uInt)max_start;
        FLUSH_BLOCK(s, 0);
    }
    /* Flush if we may have to slide, otherwise block_start may become
     * negative and the data will be gone:
     */
    if (s->strstart - (uInt)s->block_start >= MAX_DIST(s)) {
        FLUSH_BLOCK(s, 0);
    }
    FLUSH_BLOCK(s, flush == Z_FINISH);
    return flush == Z_FINISH ? finish_done : block_done;

=====
/* Compress as much as possible from the input stream, return the current
 * block state.
 * This function does not perform lazy evaluation of matches and inserts
 * new strings in the dictionary only for unmatched strings or for short
 * matches. It is used only for the fast compression options.
 */
local block_state deflate_fast(s, flush)
    deflate_state *s;
    int flush;

    IPos hash_head = NIL; /* head of the hash chain */
    int bflush;           /* set if current block must be flushed */

    for (;;) {
        /* Make sure that we always have enough lookahead, except
         * at the end of the input file. We need MAX_MATCH bytes
         * for the next match, plus MIN_MATCH bytes to insert the
         * string following the next match.
         */
        if (s->lookahead < MIN_LOOKAHEAD) {
            fill_window(s);
            if (s->lookahead < MIN_LOOKAHEAD && flush == Z_NO_FLUSH) {
                return need_more;
            }
            if (s->lookahead == 0) break; /* flush the current block */
        }

        /* Insert the string window[strstart .. strstart+2] in the
         * dictionary, and set hash_head to the head of the hash chain:
         */
        if (s->lookahead >= MIN_MATCH) {
            INSERT_STRING(s, s->strstart, hash_head);
        }

        /* Find the longest match, discarding those <= prev_length.
         * At this point we have always match_length < MIN_MATCH

```

```

    /*
    if (hash_head != NIL && s->strstart - hash_head <= MAX_DIST(s)) {
        /* To simplify the code, we prevent matches with the string
        * of window index 0 (in particular we have to avoid a match
        * of the string with itself at the start of the input file).
        */
        if (s->strategy != Z_HUFFMAN_ONLY) {
            s->match_length = longest_match(s, hash_head);
        }
        /* longest_match() sets match_start */
    }
    if (s->match_length >= MIN_MATCH) {
        check_match(s, s->strstart, s->match_start, s->match_length);

        _tr_tally_dist(s, s->strstart - s->match_start,
            s->match_length - MIN_MATCH, bflush);

        s->lookahead -= s->match_length;

        /* Insert new strings in the hash table only if the match length
        * is not too large. This saves time but degrades compression.
        */
#ifdef FASTEST
        if (s->match_length <= s->max_insert_length &&
            s->lookahead >= MIN_MATCH) {
            s->match_length--; /* string at strstart already in hash table */
            do {
                s->strstart++;
                INSERT_STRING(s, s->strstart, hash_head);
                /* strstart never exceeds WSIZE-MAX_MATCH, so there are
                * always MIN_MATCH bytes ahead.
                */
            } while (--s->match_length != 0);
            s->strstart++;
        } else
#endif
        {
            s->strstart += s->match_length;
            s->match_length = 0;
            s->ins_h = s->window[s->strstart];
            UPDATE_HASH(s, s->ins_h, s->window[s->strstart+1]);
#ifdef MIN_MATCH != 3
            Call UPDATE_HASH() MIN_MATCH-3 more times
#endif
            /* If lookahead < MIN_MATCH, ins_h is garbage, but it does not
            * matter since it will be recomputed at next deflate call.
            */
        }
    } else {
        /* No match, output a literal byte */
        Tracevv((stderr, "%c", s->window[s->strstart]));
        _tr_tally_lit(s, s->window[s->strstart], bflush);
        s->lookahead--;
        s->strstart++;
    }
    if (bflush) FLUSH_BLOCK(s, 0);
}
FLUSH_BLOCK(s, flush == Z_FINISH);
return flush == Z_FINISH ? finish_done : block_done;
}

/* =====
* Same as above, but achieves better compression. We use a lazy
* evaluation for matches: a match is finally adopted only if there is
* no better match at the next window position.
*/
local block_state deflate_slow(s, flush)
    deflate_state *s;
    int flush;
{
    IPos hash_head = NIL; /* head of hash chain */
    int bflush; /* set if current block must be flushed */

```

```

/* Process the input block
for (;;) {
    /* Make sure that we always have enough lookahead, except
    * at the end of the input file. We need MAX_MATCH bytes
    * for the next match, plus MIN_MATCH bytes to insert the
    * string following the next match.
    */
    if (s->lookahead < MIN_LOOKAHEAD) {
        fill_window(s);
        if (s->lookahead < MIN_LOOKAHEAD && flush == Z_NO_FLUSH) {
            return need_more;
        }
        if (s->lookahead == 0) break; /* flush the current block */
    }

    /* Insert the string window[strstart .. strstart+2] in the
    * dictionary, and set hash_head to the head of the hash chain:
    */
    if (s->lookahead >= MIN_MATCH) {
        INSERT_STRING(s, s->strstart, hash_head);
    }

    /* Find the longest match, discarding those <= prev_length.
    */
    s->prev_length = s->match_length, s->prev_match = s->match_start;
    s->match_length = MIN_MATCH-1;

    if (hash_head != NIL && s->prev_length < s->max_lazy_match &&
        s->strstart - hash_head <= MAX_DIST(s)) {
        /* To simplify the code, we prevent matches with the string
        * of window index 0 (in particular we have to avoid a match
        * of the string with itself at the start of the input file).
        */
        if (s->strategy != Z_HUFFMAN_ONLY) {
            s->match_length = longest_match(s, hash_head);
        }
        /* longest_match() sets match_start */

        if (s->match_length <= 5 && (s->strategy == Z_FILTERED ||
            (s->match_length == MIN_MATCH &&
            s->strstart - s->match_start > TOO_FAR))) {

            /* If prev_match is also MIN_MATCH, match_start is garbage
            * but we will ignore the current match anyway.
            */
            s->match_length = MIN_MATCH-1;
        }
    }
}

/* If there was a match at the previous step and the current
 * match is not better, output the previous match:
 */
if (s->prev_length >= MIN_MATCH && s->match_length <= s->prev_length) {
    uInt max_insert = s->strstart + s->lookahead - MIN_MATCH;
    /* Do not insert strings in hash table beyond this. */

    check_match(s, s->strstart-1, s->prev_match, s->prev_length);

    _tr_tally_dist(s, s->strstart-1 - s->prev_match,
        s->prev_length - MIN_MATCH, bflush);

    /* Insert in hash table all strings up to the end of the match.
    * strstart-1 and strstart are already inserted. If there is not
    * enough lookahead, the last two strings are not inserted in
    * the hash table.
    */
    s->lookahead -= s->prev_length-1;
    s->prev_length -= 2;
    do {
        if (++s->strstart <= max_insert) {
            INSERT_STRING(s, s->strstart, hash_head);
        }
    } while (--s->prev_length != 0);
    s->match_available = 0;
}

```



```

s->match_length = MATCH-1;
s->strstart++;

if (bflush) FLUSH_BLOCK(s, 0);

} else if (s->match_available) {
    /* If there was no match at the previous position, output a
     * single literal. If there was a match but the current match
     * is longer, truncate the previous match to a single literal.
     */
    Tracevv((stderr,"%c", s->window[s->strstart-1]));
    _tr_tally_lit(s, s->window[s->strstart-1], bflush);
    if (bflush) {
        FLUSH_BLOCK_ONLY(s, 0);
    }
    s->strstart++;
    s->lookahead--;
    if (s->strm->avail_out == 0) return need_more;
} else {
    /* There is no previous match to compare with, wait for
     * the next step to decide.
     */
    s->match_available = 1;
    s->strstart++;
    s->lookahead--;
}
}
Assert (flush != Z_NO_FLUSH, "no flush?");
if (s->match_available) {
    Tracevv((stderr,"%c", s->window[s->strstart-1]));
    _tr_tally_lit(s, s->window[s->strstart-1], bflush);
    s->match_available = 0;
}
FLUSH_BLOCK(s, flush == Z_FINISH);
return flush == Z_FINISH ? finish_done : block_done;

```

```

/* crc32.c -- compute the CRC-32 of a data stream
 * Copyright (C) 1995-1998 Mark Adler
 * For conditions of distribution and use, see copyright notice in zlib.h
 */

```

```

/* @(#) $Id$ */

```

```

#include "zlib.h"

```

```

#define local static

```

```

#ifdef DYNAMIC_CRC_TABLE

```

```

local int crc_table_empty = 1;
local uLongf crc_table[256];
local void make_crc_table OF((void));

```

```

/*
Generate a table for a byte-wise 32-bit CRC calculation on the polynomial:
 $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$ .

```

Polynomials over GF(2) are represented in binary, one bit per coefficient, with the lowest powers in the most significant bit. Then adding polynomials is just exclusive-or, and multiplying a polynomial by x is a right shift by one. If we call the above polynomial p, and represent a byte as the polynomial q, also with the lowest power in the most significant bit (so the byte 0xb1 is the polynomial x^7+x^3+x+1), then the CRC is $(q \cdot x^{32}) \bmod p$, where a mod b means the remainder after dividing a by b.

This calculation is done using the shift-register method of multiplying and taking the remainder. The register is initialized to zero, and for each incoming bit, x^{32} is added mod p to the register if the bit is a one (where $x^{32} \bmod p$ is $p+x^{32} = x^{26}+\dots+1$), and the register is multiplied mod p by x (which is shifting right by one and adding $x^{32} \bmod p$ if the bit shifted out is a one). We start with the highest power (least significant bit) of q and repeat for all eight bits of q.

The table is simply the CRC of all possible eight bit values. This is all the information needed to generate CRC's on data a byte at a time for all combinations of CRC register values and incoming bytes.

```

local void make_crc_table()

```

```

uLong c;
int n, k;
uLong poly;          /* polynomial exclusive-or pattern */
/* terms of polynomial defining this crc (except x^32): */
static const Byte p[] = {0,1,2,4,5,7,8,10,11,12,16,22,23,26};

```

```

/* make exclusive-or pattern from polynomial (0xedb88320L) */
poly = 0L;

```

```

for (n = 0; n < sizeof(p)/sizeof(Byte); n++)
    poly |= 1L << (31 - p[n]);

```

```

for (n = 0; n < 256; n++)
{
    c = (uLong)n;
    for (k = 0; k < 8; k++)
        c = c & 1 ? poly ^ (c >> 1) : c >> 1;
    crc_table[n] = c;
}

```

```

crc_table_empty = 0;

```

```

#else

```

```

/* =====
 * Table of CRC-32's of all single-byte values (made by make_crc_table)
 */

```

```

local const uLongf crc_table[256] = {
    0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL, 0x076dc419L,
    0x706af48fL, 0xe963a535L, 0x9e6495a3L, 0x0edb8832L, 0x79dcb8a4L,
    0xe0d5e91eL, 0x97d2d988L, 0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L,
    0x90b91d91L, 0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
    0x1dad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L, 0x136c9856L,

```

```

0x646ba8c0L, 0xfd62f97aL, 0x9ecL, 0x14015c4fL, 0x63066cd9L,
0xfa0f3d63L, 0x8d080df5L, 0x3b20c8L, 0x4c69105eL, 0xd56041e4L,
0xa2677172L, 0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
0x35b5a8faL, 0x42b2986cL, 0xdbb9c9d6L, 0xacbcf940L, 0x32d86ce3L,
0x45df5c75L, 0xdc6d0dcfL, 0xabd13d59L, 0x26d930acL, 0x51de003aL,
0x8d75180L, 0xbfd06116L, 0x21b4f4b5L, 0x56b3c423L, 0xcfa9599L,
0xb8bda50fL, 0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL, 0x76dc4190L,
0x01db7106L, 0x98d220bcL, 0xefd5102aL, 0x71b18589L, 0x06b6b51fL,
0x9fbfe4a5L, 0xe8b8d433L, 0x7807c9a2L, 0xf00f934L, 0x9609a88eL,
0xe10e9818L, 0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL, 0x6c0695edL,
0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L, 0x65b0d9c6L, 0x12b7e950L,
0x8bb8eb8eaL, 0xfcb9887cL, 0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L,
0xfbd44c65L, 0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL, 0x4369e96aL,
0x346ed9fcL, 0xad678846L, 0xda60b8d0L, 0x44042d73L, 0x33031de5L,
0xaa0a4c5fL, 0xdd0d7cc9L, 0x5005713cL, 0x270241aaL, 0xbe0b1010L,
0xc90c2086L, 0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L, 0x59b33d17L,
0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL, 0xedb88320L, 0x9abfb3b6L,
0x03b6e20cL, 0x74b1d29aL, 0xeada54739L, 0x9dd277afL, 0x04db2615L,
0x73dc1683L, 0xe3630b12L, 0x94643b84L, 0xd6d6a3eL, 0x7a6a5aa8L,
0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L, 0xf00f9344L,
0x8708a3d2L, 0x1e01f268L, 0x6906c2feL, 0xf762575dL, 0x806567cbL,
0x196c3671L, 0x6e6b06e7L, 0xfed41b76L, 0x89d32be0L, 0x10da7a5aL,
0x67dd4accL, 0xf9b9df6fL, 0x8eb0eeff9L, 0x17b7be43L, 0x60b08ed5L,
0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L, 0xd1bb67f1L,
0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL, 0xd80d2bdaL, 0xaf0a1b4cL,
0x36034af6L, 0x41047a60L, 0xdf60efc3L, 0xa867df55L, 0x316e8eeL,
0x4669b79L, 0xcbb1b38cL, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL, 0xc5ba3bbeL,
0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L, 0xc2d7ffa7L, 0xb5d0cf31L,
0x2cd99e8bL, 0x5bdeae1dL, 0x9b64c2b0L, 0xec63f226L, 0x756aa39cL,
0x026d930aL, 0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0xcbb61b38L, 0x92d28e9bL,
0xe5d5be0dL, 0x7cdcefb7L, 0xbdbdf21L, 0x86d3d2d4L, 0xf1d4e242L,
0x68ddb3f8L, 0x1fda036eL, 0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L,
0x18b74777L, 0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L, 0xa00ae278L,
0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L, 0xa7672661L, 0xd06016f7L,
0x4969474dL, 0x3e6e77dbL, 0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L,
0x37d83bf0L, 0xa9bcae53L, 0xdeb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L, 0xbad03605L,
0xcdd70693L, 0x54de5729L, 0x23d967bfL, 0xb3667a2eL, 0xc4614ab8L,
0x5d681b02L, 0xa6f2b94L, 0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL,
0x2d02ef8dL
#endif

/* =====
 * This function can be used by asm versions of crc32()
 */
const uLongf * ZEXPORT get_crc_table()
{
#ifdef DYNAMIC_CRC_TABLE
    if (crc_table_empty) make_crc_table();
#endif
    return (const uLongf *)crc_table;
}

/* ===== */
#define D01(buf) crc = crc_table[(((int)crc ^ (*buf++)) & 0xff) ^ (crc >> 8)];
#define D02(buf) D01(buf); D01(buf);
#define D04(buf) D02(buf); D02(buf);
#define D08(buf) D04(buf); D04(buf);

/* ===== */
uLong ZEXPORT crc32(crc, buf, len)
    uLong crc;
    const Bytef *buf;
    uInt len;
{

```

```

    if (buf == Z_NULL) return
#ifdef DYNAMIC_CRC_TABLE
    if (crc_table_empty)
        make_crc_table();
#endif
    crc = crc ^ 0xffffffffL;
    while (len >= 8)
    {
        DO8(buf);
        len -= 8;
    }
    if (len) do {
        DO1(buf);
    } while (--len);
    return crc ^ 0xffffffffL;
}

```

```

/* adler32.c -- compute the Adler-32 checksum of a data stream
 * Copyright (C) 1995-1998 Mark Adler
 * For conditions of distribution and use, see copyright notice in zlib.h
 */

/* @(#) $Id$ */

#include "zlib.h"

#define BASE 65521L /* largest prime smaller than 65536 */
#define NMAX 5552
/* NMAX is the largest n such that 255n(n+1)/2 + (n+1)(BASE-1) <= 2^32-1 */

#define DO1(buf,i)  {s1 += buf[i]; s2 += s1;}
#define DO2(buf,i)  DO1(buf,i); DO1(buf,i+1);
#define DO4(buf,i)  DO2(buf,i); DO2(buf,i+2);
#define DO8(buf,i)  DO4(buf,i); DO4(buf,i+4);
#define DO16(buf)   DO8(buf,0); DO8(buf,8);

/* ===== */
uLong ZEXPORT adler32(adler, buf, len)
    uLong adler;
    const Bytef *buf;
    ulnt len;
{
    unsigned long s1 = adler & 0xffff;
    unsigned long s2 = (adler >> 16) & 0xffff;
    int k;

    if (buf == Z_NULL) return 1L;

    while (len > 0) {
        k = len < NMAX ? len : NMAX;
        len -= k;
        while (k >= 16) {
            DO16(buf);
            buf += 16;
            k -= 16;
        }
        if (k != 0) do {
            s1 += *buf++;
            s2 += s1;
        } while (--k);
        s1 %= BASE;
        s2 %= BASE;
    }
    return (s2 << 16) | s1;
}

```

```

/* compress.c -- compress a memory buffer
 * Copyright (C) 1995-1998 Jean-loup Gailly.
 * For conditions of distribution and use, see copyright notice in zlib.h
 */

/* @(#) $Id$ */

#include "zlib.h"

/* =====
Compresses the source buffer into the destination buffer. The level
parameter has the same meaning as in deflateInit. sourceLen is the byte
length of the source buffer. Upon entry, destLen is the total size of the
destination buffer, which must be at least 0.1% larger than sourceLen plus
12 bytes. Upon exit, destLen is the actual size of the compressed buffer.

compress2 returns Z_OK if success, Z_MEM_ERROR if there was not enough
memory, Z_BUF_ERROR if there was not enough room in the output buffer,
Z_STREAM_ERROR if the level parameter is invalid.
*/
int ZEXPORT compress2 (dest, destLen, source, sourceLen, level)
    Bytef *dest;
    uLongf *destLen;
    const Bytef *source;
    uLong sourceLen;
    int level;
{
    z_stream stream;
    int err;

    stream.next_in = (Bytef*)source;
    stream.avail_in = (uInt)sourceLen;
#ifdef MAXSEG_64K
    /* Check for source > 64K on 16-bit machine: */
    if ((uLong)stream.avail_in != sourceLen) return Z_BUF_ERROR;
#endif
    stream.next_out = dest;
    stream.avail_out = (uInt)*destLen;
    if ((uLong)stream.avail_out != *destLen) return Z_BUF_ERROR;

    stream.zalloc = (alloc_func)0;
    stream.zfree = (free_func)0;
    stream.opaque = (voidpf)0;

    err = deflateInit(&stream, level);
    if (err != Z_OK) return err;

    err = deflate(&stream, Z_FINISH);
    if (err != Z_STREAM_END) {
        deflateEnd(&stream);
        return err == Z_OK ? Z_BUF_ERROR : err;
    }
    *destLen = stream.total_out;

    err = deflateEnd(&stream);
    return err;
}

/* =====
*/
int ZEXPORT compress (dest, destLen, source, sourceLen)
    Bytef *dest;
    uLongf *destLen;
    const Bytef *source;
    uLong sourceLen;
{
    return compress2(dest, destLen, source, sourceLen, Z_DEFAULT_COMPRESSION);
}

```

```
/* gzio.c -- IO on .gz files
 * Copyright (C) 1995-1998 Jean-loup Gailly.
 * For conditions of distribution and use, see copyright notice in
 * zlib.h
 *
 * Compile this file with -DNO_DEFLATE to avoid the compression co
 * de.
 */

/* @(#) $Id$ */

#include <stdio.h>

#include "zutil.h"

struct internal_state {int dummy;}; /* for buggy compilers */

#ifndef Z_BUFSIZE
# ifdef MAXSEG_64K
#   define Z_BUFSIZE 4096 /* minimize memory usage for 16-bit DOS
 */
# else
#   define Z_BUFSIZE 16384
# endif
#endif
#ifndef Z_PRINTF_BUFSIZE
# define Z_PRINTF_BUFSIZE 4096
#endif

#define ALLOC(size) malloc(size)
#define TRYFREE(p) {if (p) free(p);}

static int gz_magic[2] = {0x1f, 0x8b}; /* gzip magic header */

/* gzip flag byte */
#define ASCII_FLAG      0x01 /* bit 0 set: file probably ascii text */
/
#define HEAD_CRC        0x02 /* bit 1 set: header CRC present */
#define EXTRA_FIELD     0x04 /* bit 2 set: extra field present */
#define ORIG_NAME       0x08 /* bit 3 set: original file name present
 */
#define COMMENT         0x10 /* bit 4 set: file comment present */
#define RESERVED        0xE0 /* bits 5..7: reserved */

typedef struct gz_stream {
    z_stream stream;
```

```

    int      z_err;    /* error code for last stream operation */
    int      z_eof;    /* set if end of input file */
    FILE     *file;    /* .gz file */
    Byte     *inbuf;   /* input buffer */
    Byte     *outbuf;  /* output buffer */
    uLong    crc;      /* crc32 of uncompressed data */
    char     *msg;     /* error message */
    char     *path;    /* path name for debugging only */
    int      transparent; /* 1 if input file is not a .gz file */
    char     mode;     /* 'w' or 'r' */
    long     startpos; /* start of compressed data in file (header
skipped) */
} gz_stream;

```

```

local gzFile gz_open      OF((const char *path, const char *mode,
int fd));
local int do_flush      OF((gzFile file, int flush));
local int get_byte      OF((gz_stream *s));
local void check_header OF((gz_stream *s));
local int destroy       OF((gz_stream *s));
local void putLong      OF((FILE *file, uLong x));
local uLong getLong     OF((gz_stream *s));

```

```

/* =====
=====

```

Opens a gzip (.gz) file for reading or writing. The mode parameter

is as in fopen ("rb" or "wb"). The file is given either by file descriptor

or path name (if fd == -1).

gz_open return NULL if the file could not be opened or if there was

insufficient memory to allocate the (de)compression state; errno

can be checked to distinguish the two cases (if errno is zero, the

zlib error is Z_MEM_ERROR).

*/

```

local gzFile gz_open (path, mode, fd)

```

```

    const char *path;

```

```

    const char *mode;

```

```

    int fd;

```

```

{

```

```

    int err;

```

```

    int level = Z_DEFAULT_COMPRESSION; /* compression level */

```



```

int strategy = Z_DEFAULT_STRATEGY; /* compression strategy */
char *p = (char*)mode;
gz_stream *s;
char fmode[80]; /* copy of mode, without the compression level
*/
char *m = fmode;

if (!path || !mode) return Z_NULL;

s = (gz_stream *)ALLOC(sizeof(gz_stream));
if (!s) return Z_NULL;

s->stream.zalloc = (alloc_func)0;
s->stream.zfree = (free_func)0;
s->stream.opaque = (voidpf)0;
s->stream.next_in = s->inbuf = Z_NULL;
s->stream.next_out = s->outbuf = Z_NULL;
s->stream.avail_in = s->stream.avail_out = 0;
s->file = NULL;
s->z_err = Z_OK;
s->z_eof = 0;
s->crc = crc32(0L, Z_NULL, 0);
s->msg = NULL;
s->transparent = 0;

s->path = (char*)ALLOC(strlen(path)+1);
if (s->path == NULL) {
    return destroy(s), (gzFile)Z_NULL;
}
strcpy(s->path, path); /* do this early for debugging */

s->mode = '\0';
do {
    if (*p == 'r') s->mode = 'r';
    if (*p == 'w' || *p == 'a') s->mode = 'w';
    if (*p >= '0' && *p <= '9') {
        level = *p - '0';
    } else if (*p == 'f') {
        strategy = Z_FILTERED;
    } else if (*p == 'h') {
        strategy = Z_HUFFMAN_ONLY;
    } else {
        *m++ = *p; /* copy the mode */
    }
} while (*p++ && m != fmode + sizeof(fmode));
if (s->mode == '\0') return destroy(s), (gzFile)Z_NULL;

```

```

    if (s->mode == 'w') {
#ifdef NO_DEFLATE
        err = Z_STREAM_ERROR;
#else
        err = deflateInit2(&(s->stream), level,
                           Z_DEFLATED, -MAX_WBITS, DEF_MEM_LEVEL,
strategy);
        /* windowBits is passed < 0 to suppress zlib header */

        s->stream.next_out = s->outbuf = (Byte*)ALLOC(Z_BUFSIZE);
#endif
        if (err != Z_OK || s->outbuf == Z_NULL) {
            return destroy(s), (gzFile)Z_NULL;
        }
    } else {
        s->stream.next_in = s->inbuf = (Byte*)ALLOC(Z_BUFSIZE);

        err = inflateInit2(&(s->stream), -MAX_WBITS);
        /* windowBits is passed < 0 to tell that there is no zlib
header.
        * Note that in this case inflate *requires* an extra "dum
my" byte
        * after the compressed stream in order to complete decomp
ression and
        * return Z_STREAM_END. Here the gzip CRC32 ensures that 4
bytes are
        * present after the compressed stream.
        */
        if (err != Z_OK || s->inbuf == Z_NULL) {
            return destroy(s), (gzFile)Z_NULL;
        }
    }
    s->stream.avail_out = Z_BUFSIZE;

    errno = 0;
    s->file = fd < 0 ? F_OPEN(path, fmode) : (FILE*)fdopen(fd, fmo
de);

    if (s->file == NULL) {
        return destroy(s), (gzFile)Z_NULL;
    }
    if (s->mode == 'w') {
        /* Write a very simple .gz header:
        */
        fprintf(s->file, "%c%c%c%c%c%c%c%c%c%c", gz_magic[0], gz_m

```

```

agic[1],
        Z_DEFLATED, 0 /*flags*/, 0,0,0,0 /*time*/, 0 /*xflags
*/, OS_CODE);
    s->startpos = 10L;
    /* We use 10L instead of ftell(s->file) to because ftell c
auses an
        * fflush on some systems. This version of the library doe
sn't use
        * startpos anyway in write mode, so this initialization i
s not
        * necessary.
    */
    } else {
        check_header(s); /* skip the .gz header */
        s->startpos = (ftell(s->file) - s->stream.avail_in);
    }

    return (gzFile)s;
}

/* =====
=====
    Opens a gzip (.gz) file for reading or writing.
*/
gzFile ZEXPORT gzopen (path, mode)
    const char *path;
    const char *mode;
{
    return gz_open (path, mode, -1);
}

/* =====
=====
    Associate a gzFile with the file descriptor fd. fd is not dup
'ed here
    to mimic the behavio(u)r of fdopen.
*/
gzFile ZEXPORT gzdopen (fd, mode)
    int fd;
    const char *mode;
{
    char name[20];

    if (fd < 0) return (gzFile)Z_NULL;
    sprintf(name, "<fd:%d>", fd); /* for debugging */

```

```

    return gz_open (name, mode, fd);
}

/* =====
=====
* Update the compression level and strategy
*/
int ZEXPORT gzsetparams (file, level, strategy)
    gzFile file;
    int level;
    int strategy;
{
    gz_stream *s = (gz_stream*)file;

    if (s == NULL || s->mode != 'w') return Z_STREAM_ERROR;

    /* Make room to allow flushing */
    if (s->stream.avail_out == 0) {

        s->stream.next_out = s->outbuf;
        if (fwrite(s->outbuf, 1, Z_BUFSIZE, s->file) != Z_BUFSIZE)
        {
            s->z_err = Z_ERRNO;
        }
        s->stream.avail_out = Z_BUFSIZE;
    }

    return deflateParams (&(s->stream), level, strategy);
}

/* =====
=====
    Read a byte from a gz_stream; update next_in and avail_in. Re-
turn EOF
    for end of file.
    IN assertion: the stream s has been successfully opened for read-
ing.
*/
local int get_byte(s)
    gz_stream *s;
{
    if (s->z_eof) return EOF;
    if (s->stream.avail_in == 0) {
        errno = 0;
        s->stream.avail_in = fread(s->inbuf, 1, Z_BUFSIZE, s->file
);
    }

```

```

        if (s->stream.avail_in == 0) {
            s->z_eof = 1;
            if (ferror(s->file)) s->z_err = Z_ERRNO;
            return EOF;
        }
        s->stream.next_in = s->inbuf;
    }
    s->stream.avail_in--;
    return *(s->stream.next_in)++;
}

/* =====
=====
    Check the gzip header of a gz_stream opened for reading. Set
    the stream
    mode to transparent if the gzip magic header is not present; s
    et s->err
    to Z_DATA_ERROR if the magic header is present but the rest of
    the header
    is incorrect.
    IN assertion: the stream s has already been created sucessfull
    y;
    s->stream.avail_in is zero for the first time, but may be n
    on-zero
    for concatenated .gz files.
*/
local void check_header(s)
    gz_stream *s;
{
    int method; /* method byte */
    int flags; /* flags byte */
    uInt len;
    int c;

    /* Check the gzip magic header */
    for (len = 0; len < 2; len++) {
        c = get_byte(s);
        if (c != gz_magic[len]) {
            if (len != 0) s->stream.avail_in++, s->stream.next_in-
-;
            if (c != EOF) {
                s->stream.avail_in++, s->stream.next_in--;
                s->transparent = 1;
            }
            s->z_err = s->stream.avail_in != 0 ? Z_OK : Z_STREAM_E
ND;

```

```

        return;
    }
}
method = get_byte(s);
flags = get_byte(s);
if (method != Z_DEFLATED || (flags & RESERVED) != 0) {
    s->z_err = Z_DATA_ERROR;
    return;
}

/* Discard time, xflags and OS code: */
for (len = 0; len < 6; len++) (void)get_byte(s);

if ((flags & EXTRA_FIELD) != 0) { /* skip the extra field */
    len = (uInt)get_byte(s);
    len += ((uInt)get_byte(s))<<8;
    /* len is garbage if EOF but the loop below will quit anyw
ay */
    while (len-- != 0 && get_byte(s) != EOF) ;
}
if ((flags & ORIG_NAME) != 0) { /* skip the original file name
*/
    while ((c = get_byte(s)) != 0 && c != EOF) ;
}
if ((flags & COMMENT) != 0) { /* skip the .gz file comment */
/
    while ((c = get_byte(s)) != 0 && c != EOF) ;
}
if ((flags & HEAD_CRC) != 0) { /* skip the header crc */
    for (len = 0; len < 2; len++) (void)get_byte(s);
}
s->z_err = s->z_eof ? Z_DATA_ERROR : Z_OK;
}

/* =====
=====
* Cleanup then free the given gz_stream. Return a zlib error code
.
* Try freeing in the reverse order of allocations.
*/
local int destroy (s)
    gz_stream *s;
{
    int err = Z_OK;

    if (!s) return Z_STREAM_ERROR;

```

```

    TRYFREE(s->msg);

    if (s->stream.state != NULL) {
        if (s->mode == 'w') {
#ifdef NO_DEFLATE
            err = Z_STREAM_ERROR;
#else
            err = deflateEnd(&(s->stream));
#endif
        } else if (s->mode == 'r') {
            err = inflateEnd(&(s->stream));
        }
    }
    if (s->file != NULL && fclose(s->file)) {
#ifdef ESPIPE
        if (errno != ESPIPE) /* fclose is broken for pipes in HP/U
X */
#endif
        err = Z_ERRNO;
    }
    if (s->z_err < 0) err = s->z_err;

    TRYFREE(s->inbuf);
    TRYFREE(s->outbuf);
    TRYFREE(s->path);
    TRYFREE(s);
    return err;
}

/* =====
=====
    Reads the given number of uncompressed bytes from the compressed file.
    gzread returns the number of bytes actually read (0 for end of file).
*/
int ZEXPORT gzread (file, buf, len)
    gzFile file;
    voidp buf;
    unsigned len;
{
    gz_stream *s = (gz_stream*)file;
    Bytef *start = (Bytef*)buf; /* starting point for crc computation */
    Byte *next_out; /* == stream.next_out but not forced far (for

```

MSDOS) */

if (s == NULL || s->mode != 'r') return Z_STREAM_ERROR;

if (s->z_err == Z_DATA_ERROR || s->z_err == Z_ERRNO) return -1

;

if (s->z_err == Z_STREAM_END) return 0; /* EOF */

next_out = (Byte*)buf;

s->stream.next_out = (Bytef*)buf;

s->stream.avail_out = len;

while (s->stream.avail_out != 0) {

if (s->transparent) {

/* Copy first the lookahead bytes: */

uInt n = s->stream.avail_in;

if (n > s->stream.avail_out) n = s->stream.avail_out;

if (n > 0) {

zmemcpy(s->stream.next_out, s->stream.next_in, n);

next_out += n;

s->stream.next_out = next_out;

s->stream.next_in += n;

s->stream.avail_out -= n;

s->stream.avail_in -= n;

}

if (s->stream.avail_out > 0) {

s->stream.avail_out -= fread(next_out, 1, s->stream.avail_out,

m.avail_out,

s->file);

}

len -= s->stream.avail_out;

s->stream.total_in += (uLong)len;

s->stream.total_out += (uLong)len;

if (len == 0) s->z_eof = 1;

return (int)len;

}

if (s->stream.avail_in == 0 && !s->z_eof) {

errno = 0;

s->stream.avail_in = fread(s->inbuf, 1, Z_BUF_SIZE, s->

file);

if (s->stream.avail_in == 0) {

s->z_eof = 1;

if (ferror(s->file)) {

s->z_err = Z_ERRNO;


```

        break;
    }
}
s->stream.next_in = s->inbuf;
}
s->z_err = inflate(&(s->stream), Z_NO_FLUSH);

if (s->z_err == Z_STREAM_END) {
    /* Check CRC and original size */
    s->crc = crc32(s->crc, start, (uInt)(s->stream.next_out
t - start));
    start = s->stream.next_out;

    if (getLong(s) != s->crc) {
        s->z_err = Z_DATA_ERROR;
    } else {
        (void)getLong(s);
        /* The uncompressed length returned by above getlo
ng() may
of
        * be different from s->stream.total_out) in case
        * concatenated .gz files. Check for such files:
        */
        check_header(s);
        if (s->z_err == Z_OK) {
            uLong total_in = s->stream.total_in;
            uLong total_out = s->stream.total_out;

            inflateReset(&(s->stream));
            s->stream.total_in = total_in;
            s->stream.total_out = total_out;
            s->crc = crc32(0L, Z_NULL, 0);
        }
    }
}
if (s->z_err != Z_OK || s->z_eof) break;
}
s->crc = crc32(s->crc, start, (uInt)(s->stream.next_out - star
t));

return (int)(len - s->stream.avail_out);
}

```

```

/* =====
=====

```

Reads one byte from the compressed file. gzgetc returns this byte or -1 in case of end of file or error.

```
*/
int ZEXPORT gzgetc(file)
    gzFile file;
{
    unsigned char c;

    return gzread(file, &c, 1) == 1 ? c : -1;
}
```

```
/* =====
=====
```

Reads bytes from the compressed file until len-1 characters are read, or a newline character is read and transferred to buf, or an end-of-file condition is encountered. The string is then terminated with a null character. gzgets returns buf, or Z_NULL in case of error.

The current implementation is not optimized at all.

```
*/
char * ZEXPORT gzgets(file, buf, len)
    gzFile file;
    char *buf;
    int len;
{
    char *b = buf;
    if (buf == Z_NULL || len <= 0) return Z_NULL;

    while (--len > 0 && gzread(file, buf, 1) == 1 && *buf++ != '\n') ;
    *buf = '\0';
    return b == buf && len > 0 ? Z_NULL : b;
}
```

```
#ifndef NO_DEFLATE
```

```
/* =====
=====
```

Writes the given number of uncompressed bytes into the compressed file.

gzwrite returns the number of bytes actually written (0 in case of error).

*/

```
int ZEXPORT gzwrite (file, buf, len)
    gzFile file;
    const voidp buf;
    unsigned len;
{
    gz_stream *s = (gz_stream*)file;

    if (s == NULL || s->mode != 'w') return Z_STREAM_ERROR;

    s->stream.next_in = (Bytef*)buf;
    s->stream.avail_in = len;

    while (s->stream.avail_in != 0) {
        if (s->stream.avail_out == 0) {
            s->stream.next_out = s->outbuf;
            if (fwrite(s->outbuf, 1, Z_BUF_SIZE, s->file) != Z_BUF_S
IZE) {
                s->z_err = Z_ERRNO;
                break;
            }
            s->stream.avail_out = Z_BUF_SIZE;
        }
        s->z_err = deflate(&(s->stream), Z_NO_FLUSH);
        if (s->z_err != Z_OK) break;
    }
    s->crc = crc32(s->crc, (const Bytef *)buf, len);

    return (int)(len - s->stream.avail_in);
}
```

/* =====

Converts, formats, and writes the args to the compressed file under control of the format string, as in fprintf. gzprintf returns the number of uncompressed bytes actually written (0 in case of error).

*/

```
#ifdef STDC
#include <stdarg.h>
```

```

int ZEXPORTVA gzprintf (gzFile file, const char *format, /* args *
/ ...)
{
    char buf[Z_PRINTF_BUFSIZE];
    va_list va;
    int len;

    va_start(va, format);
#ifdef HAS_vsnprintf
    (void)vsnprintf(buf, sizeof(buf), format, va);
#else
    (void)vsprintf(buf, format, va);
#endif
    va_end(va);
    len = strlen(buf); /* some *sprintf don't return the nb of byt
es written */
    if (len <= 0) return 0;

    return gzwrite(file, buf, (unsigned)len);
}
#else /* not ANSI C */

int ZEXPORTVA gzprintf (file, format, a1, a2, a3, a4, a5, a6, a7,
a8, a9, a10,
                        a11, a12, a13, a14, a15, a16, a17, a18, a19
, a20)
    gzFile file;
    const char *format;
    int a1, a2, a3, a4, a5, a6, a7, a8, a9, a10,
        a11, a12, a13, a14, a15, a16, a17, a18, a19, a20;
{
    char buf[Z_PRINTF_BUFSIZE];
    int len;

#ifdef HAS_snprintf
    snprintf(buf, sizeof(buf), format, a1, a2, a3, a4, a5, a6, a7,
a8,
                a9, a10, a11, a12, a13, a14, a15, a16, a17, a18, a19,
a20);
#else
    sprintf(buf, format, a1, a2, a3, a4, a5, a6, a7, a8,
                a9, a10, a11, a12, a13, a14, a15, a16, a17, a18, a19,
a20);
#endif
    len = strlen(buf); /* old sprintf doesn't return the nb of byt
es written */

```

```

    if (len <= 0) return 0;

    return gzwrite(file, buf, len);
}
#endif

/* =====
   Writes c, converted to an unsigned char, into the compressed
   file.
   gzputc returns the value that was written, or -1 in case of err
   or.
   */
int ZEXPORT gzputc(file, c)
    gzFile file;
    int c;
{
    unsigned char cc = (unsigned char) c; /* required for big endi
    an systems */

    return gzwrite(file, &cc, 1) == 1 ? (int)cc : -1;
}

/* =====
   Writes the given null-terminated string to the compressed fi
   le, excluding
   the terminating null character.
   gzputs returns the number of characters written, or -1 in ca
   se of error.
   */
int ZEXPORT gzputs(file, s)
    gzFile file;
    const char *s;
{
    return gzwrite(file, (char*)s, (unsigned)strlen(s));
}

/* =====
   Flushes all pending output into the compressed file. The para
   meter
   flush is as in the deflate() function.
   */

```

```

local int do_flush (file, flush)
    gzFile file;
    int flush;
{
    uInt len;
    int done = 0;
    gz_stream *s = (gz_stream*)file;

    if (s == NULL || s->mode != 'w') return Z_STREAM_ERROR;

    s->stream.avail_in = 0; /* should be zero already anyway */

    for (;;) {
        len = Z_BUF_SIZE - s->stream.avail_out;

        if (len != 0) {
            if ((uInt)fwrite(s->outbuf, 1, len, s->file) != len) {
                s->z_err = Z_ERRNO;
                return Z_ERRNO;
            }
            s->stream.next_out = s->outbuf;
            s->stream.avail_out = Z_BUF_SIZE;
        }
        if (done) break;
        s->z_err = deflate(&(s->stream), flush);

        /* Ignore the second of two consecutive flushes: */
        if (len == 0 && s->z_err == Z_BUF_ERROR) s->z_err = Z_OK;

        /* deflate has finished flushing only when it hasn't used
up
        * all the available space in the output buffer:
        */
        done = (s->stream.avail_out != 0 || s->z_err == Z_STREAM_E
ND);

        if (s->z_err != Z_OK && s->z_err != Z_STREAM_END) break;
    }
    return s->z_err == Z_STREAM_END ? Z_OK : s->z_err;
}

int ZEXPORT gzflush (file, flush)
    gzFile file;
    int flush;
{
    gz_stream *s = (gz_stream*)file;

```

```

    int err = do_flush (file, flush);

    if (err) return err;
    fflush(s->file);
    return s->z_err == Z_STREAM_END ? Z_OK : s->z_err;
}
#endif /* NO_DEFLATE */

/* =====
=====
    Sets the starting position for the next gzread or gzwrite on
    the given
    compressed file. The offset represents a number of bytes in the
    gzseek returns the resulting offset location as measured in
    bytes from
    the beginning of the uncompressed stream, or -1 in case of erro
    r.

    SEEK_END is not implemented, returns error.
    In this version of the library, gzseek can be extremely slow
    .
    */
z_off_t ZEXPORT gzseek (file, offset, whence)
    gzFile file;
    z_off_t offset;
    int whence;
{
    gz_stream *s = (gz_stream*)file;

    if (s == NULL || whence == SEEK_END ||
        s->z_err == Z_ERRNO || s->z_err == Z_DATA_ERROR) {
        return -1L;
    }

    if (s->mode == 'w') {
#ifdef NO_DEFLATE
        return -1L;
#else
        if (whence == SEEK_SET) {
            offset -= s->stream.total_in;
        }
        if (offset < 0) return -1L;

        /* At this point, offset is the number of zero bytes to wr
ite. */
        if (s->inbuf == Z_NULL) {
            s->inbuf = (Byte*)ALLOC(Z_BUFSIZE); /* for seeking */

```

```

        zmemzero(s->inbuf, Z_BUFSIZE);
    }
    while (offset > 0) {
        uInt size = Z_BUFSIZE;
        if (offset < Z_BUFSIZE) size = (uInt)offset;

        size = gzwrite(file, s->inbuf, size);
        if (size == 0) return -1L;

        offset -= size;
    }
    return (z_off_t)s->stream.total_in;
#endif
}
/* Rest of function is for reading only */

/* compute absolute position */
if (whence == SEEK_CUR) {
    offset += s->stream.total_out;
}
if (offset < 0) return -1L;

if (s->transparent) {
    /* map to fseek */
    s->stream.avail_in = 0;
    s->stream.next_in = s->inbuf;
    if (fseek(s->file, offset, SEEK_SET) < 0) return -1L;

    s->stream.total_in = s->stream.total_out = (uLong)offset;
    return offset;
}

/* For a negative seek, rewind and use positive seek */
if ((uLong)offset >= s->stream.total_out) {
    offset -= s->stream.total_out;
} else if (gzrewind(file) < 0) {
    return -1L;
}
/* offset is now the number of bytes to skip. */

if (offset != 0 && s->outbuf == Z_NULL) {
    s->outbuf = (Byte*)ALLOC(Z_BUFSIZE);
}
while (offset > 0) {
    int size = Z_BUFSIZE;
    if (offset < Z_BUFSIZE) size = (int)offset;

```



```

        size = gzread(file, s->outbuf, (uInt)size);
        if (size <= 0) return -1L;
        offset -= size;
    }
    return (z_off_t)s->stream.total_out;
}

/* =====
=====
    Rewinds input file.
*/
int ZEXPORT gzrewind (file)
    gzFile file;
{
    gz_stream *s = (gz_stream*)file;

    if (s == NULL || s->mode != 'r') return -1;

    s->z_err = Z_OK;
    s->z_eof = 0;
    s->stream.avail_in = 0;
    s->stream.next_in = s->inbuf;
    s->crc = crc32(0L, Z_NULL, 0);

    if (s->startpos == 0) { /* not a compressed file */
        rewind(s->file);
        return 0;
    }

    (void) inflateReset(&s->stream);
    return fseek(s->file, s->startpos, SEEK_SET);
}

/* =====
=====
    Returns the starting position for the next gzread or gzwrite
    on the
    given compressed file. This position represents a number of byt
    es in the
    uncompressed data stream.
*/
z_off_t ZEXPORT gztell (file)
    gzFile file;
{
    return gzseek(file, 0L, SEEK_CUR);
}

```

```

}

/* =====
Returns 1 when EOF has previously been detected reading the g
iven
input stream, otherwise zero.
*/
int ZEXPORT gzeof (file)
    gzFile file;
{
    gz_stream *s = (gz_stream*)file;

    return (s == NULL || s->mode != 'r') ? 0 : s->z_eof;
}

/* =====
Outputs a long in LSB order to the given file
*/
local void putLong (file, x)
    FILE *file;
    uLong x;
{
    int n;
    for (n = 0; n < 4; n++) {
        fputc((int)(x & 0xff), file);
        x >>= 8;
    }
}

/* =====
Reads a long in LSB order from the given gz_stream. Sets z_err
in case
of error.
*/
local uLong getLong (s)
    gz_stream *s;
{
    uLong x = (uLong)get_byte(s);
    int c;

    x += ((uLong)get_byte(s))<<8;
    x += ((uLong)get_byte(s))<<16;
    c = get_byte(s);

```

```

    if (c == EOF) s->z_err = Z_DATA_ERROR;
    x += ((uLong)c)<<24;
    return x;
}

/* =====
=====
    Flushes all pending output if necessary, closes the compressed
    file
    and deallocates all the (de)compression state.
*/
int ZEXPORT gzclose (file)
    gzFile file;
{
    int err;
    gz_stream *s = (gz_stream*)file;

    if (s == NULL) return Z_STREAM_ERROR;

    if (s->mode == 'w') {
#ifdef NO_DEFLATE
        return Z_STREAM_ERROR;
#else
        err = do_flush (file, Z_FINISH);
        if (err != Z_OK) return destroy((gz_stream*)file);

        putLong (s->file, s->crc);
        putLong (s->file, s->stream.total_in);
#endif
    }
    return destroy((gz_stream*)file);
}

/* =====
=====
    Returns the error message for the last error which occurred on
    the
    given compressed file. errnum is set to zlib error number. If a
    n
    error occurred in the file system and not in the compression library,
    errnum is set to Z_ERRNO and the application may consult errno
    to get the exact error code.
*/
const char* ZEXPORT gzerror (file, errnum)
    gzFile file;

```

```

int *errnum;
{
    char *m;
    gz_stream *s = (gz_stream*)file;

    if (s == NULL) {
        *errnum = Z_STREAM_ERROR;
        return (const char*)ERR_MSG(Z_STREAM_ERROR);
    }
    *errnum = s->z_err;
    if (*errnum == Z_OK) return (const char*)"";

    m = (char*)(*errnum == Z_ERRNO ? zstrerror(errno) : s->stream
.msg);

    if (m == NULL || *m == '\\0') m = (char*)ERR_MSG(s->z_err);

    TRYFREE(s->msg);
    s->msg = (char*)ALLOC(strlen(s->path) + strlen(m) + 3);
    strcpy(s->msg, s->path);
    strcat(s->msg, ": ");
    strcat(s->msg, m);
    return (const char*)s->msg;
}

```

```

/* infblock.c -- interpret and access block types to last block
 * Copyright (C) 1995-1998 Mark Adler
 * For conditions of distribution and use, see copyright notice in zlib.h
 */

```

```

#include "zutil.h"
#include "infblock.h"
#include "inftrees.h"
#include "infcodes.h"
#include "infutil.h"

```

```

struct inflate_codes_state {int dummy;}; /* for buggy compilers */

```

```

/* simplify the use of the inflate_huft type with some defines */

```

```

#define exop word.what.Exop
#define bits word.what.Bits

```

```

/* Table for deflate from PKZIP's appnote.txt. */

```

```

local const uint border[] = { /* Order of the bit length code lengths */
    16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15};

```

```

/*

```

Notes beyond the 1.93a appnote.txt:

1. Distance pointers never point before the beginning of the output stream.
2. Distance pointers can point back across blocks, up to 32k away.
3. There is an implied maximum of 7 bits for the bit length table and 15 bits for the actual data.
4. If only one code exists, then it is encoded using one bit. (Zero would be more efficient, but perhaps a little confusing.) If two codes exist, they are coded using one bit each (0 and 1).
5. There is no way of sending zero distance codes--a dummy must be sent if there are none. (History: a pre 2.0 version of PKZIP would store blocks with no distance codes, but this was discovered to be too harsh a criterion.) Valid only for 1.93a. 2.04c does allow zero distance codes, which is sent as one code of zero bits in length.
6. There are up to 286 literal/length codes. Code 256 represents the end-of-block. Note however that the static length tree defines 288 codes just to fill out the Huffman codes. Codes 286 and 287 cannot be used though, since there is no length base or extra bits defined for them. Similarly, there are up to 30 distance codes. However, static trees define 32 codes (all 5 bits) to fill out the Huffman codes, but the last two had better not show up in the data.
7. Unzip can check dynamic Huffman blocks for complete code sets. The exception is that a single code would not be complete (see #4).
8. The five bits following the block type is really the number of literal codes sent minus 257.
9. Length codes 8,16,16 are interpreted as 13 length codes of 8 bits (1+6+6). Therefore, to output three times the length, you output three codes (1+1+1), whereas to output four times the same length, you only need two codes (1+3). Hmm.
10. In the tree reconstruction algorithm, Code = Code + Increment only if BitLength(i) is not zero. (Pretty obvious.)
11. Correction: 4 Bits: # of Bit Length codes - 4 (4 - 19)
12. Note: length code 284 can represent 227-258, but length code 285 really is 258. The last length deserves its own, short code since it gets used a lot in very redundant files. The length 258 is special since 258 - 3 (the min match length) is 255.
13. The literal/length and distance code bit lengths are read as a single stream of lengths. It is possible (and advantageous) for a repeat code (16, 17, or 18) to go across the boundary between the two sets of lengths.

```

*/

```

```

void inflate_blocks_reset(s, z, c)
inflate_blocks_statef *s;
z_stream *z;
uLongf *c;
{
    if (c != Z_NULL)

```

```

    *c = s->check;
    if (s->mode == BTREE || s->mode == DTREE)
        ZFREE(z, s->sub.trees.blens);
    if (s->mode == CODES)
        inflate_codes_free(s->sub.decode.codes, z);
    s->mode = TYPE;
    s->bitk = 0;
    s->bitb = 0;
    s->read = s->write = s->window;
    if (s->checkfn != Z_NULL)
        z->adler = s->check = (*s->checkfn)(0L, (const Bytef *)Z_NULL, 0);
    Tracev((stderr, "inflate:   blocks reset\n"));
}

```

```

inflate_blocks_statef *inflate_blocks_new(z, c, w)
Z_streamf z;
check_func c;
uInt w;
{
    inflate_blocks_statef *s;

    if ((s = (inflate_blocks_statef *)ZALLOC
        (z, 1, sizeof(struct inflate_blocks_state))) == Z_NULL)
        return s;
    if ((s->hufts =
        (inflate_huft *)ZALLOC(z, sizeof(inflate_huft), MANY)) == Z_NULL)
    {
        ZFREE(z, s);
        return Z_NULL;
    }
    if ((s->window = (Bytef *)ZALLOC(z, 1, w)) == Z_NULL)
    {
        ZFREE(z, s->hufts);
        ZFREE(z, s);
        return Z_NULL;
    }
    s->end = s->window + w;
    s->checkfn = c;
    s->mode = TYPE;
    Tracev((stderr, "inflate:   blocks allocated\n"));
    inflate_blocks_reset(s, z, Z_NULL);
    return s;
}

```

```

int inflate_blocks(s, z, r)
inflate_blocks_statef *s;
Z_streamf z;
int r;
{
    uInt t;          /* temporary storage */
    uLong b;         /* bit buffer */
    uInt k;          /* bits in bit buffer */
    Bytef *p;        /* input data pointer */
    uInt n;          /* bytes available there */
    Bytef *q;        /* output window write pointer */
    uInt m;          /* bytes to end of window or read pointer */

    /* copy input/output information to locals (UPDATE macro restores) */
    LOAD

    /* process input based on current state */
    while (1) switch (s->mode)
    {
        case TYPE:
            NEEDBITS(3)
            t = (uInt)b & 7;
            s->last = t & 1;
            switch (t >> 1)
            {
                case 0:
                    /* stored */
                    Tracev((stderr, "inflate:   stored block%s\n",

```

```

        s->last ? " (" : "");
DUMPBITS(3)
t = k & 7; /* go to byte boundary */
DUMPBITS(t)
s->mode = LENS; /* get length of stored block */
break;
case 1: /* fixed */
    Tracev((stderr, "inflate: fixed codes block%s\n",
        s->last ? " (last)" : ""));
    {
        uInt bl, bd;
        inflate_huft *tl, *td;

        inflate_trees_fixed(&bl, &bd, &tl, &td, z);
        s->sub.decode.codes = inflate_codes_new(bl, bd, tl, td, z);
        if (s->sub.decode.codes == Z_NULL)
        {
            r = Z_MEM_ERROR;
            LEAVE
        }
    }
    DUMPBITS(3)
    s->mode = CODES;
    break;
case 2: /* dynamic */
    Tracev((stderr, "inflate: dynamic codes block%s\n",
        s->last ? " (last)" : ""));
    DUMPBITS(3)
    s->mode = TABLE;
    break;
case 3: /* illegal */
    DUMPBITS(3)
    s->mode = BAD;
    z->msg = (char*)"invalid block type";
    r = Z_DATA_ERROR;
    LEAVE
}
break;
case LENS:
    NEEDBITS(32)
    if (((b >> 16) & 0xffff) != (b & 0xffff))
    {
        s->mode = BAD;
        z->msg = (char*)"invalid stored block lengths";
        r = Z_DATA_ERROR;
        LEAVE
    }
    s->sub.left = (uInt)b & 0xffff;
    b = k = 0; /* dump bits */
    Tracev((stderr, "inflate: stored length %u\n", s->sub.left));
    s->mode = s->sub.left ? STORED : (s->last ? DRY : TYPE);
    break;
case STORED:
    if (n == 0)
        LEAVE
    NEEDOUT
    t = s->sub.left;
    if (t > n) t = n;
    if (t > m) t = m;
    zmemcpy(q, p, t);
    p += t; n -= t;
    q += t; m -= t;
    if ((s->sub.left -= t) != 0)
        break;
    Tracev((stderr, "inflate: stored end, %lu total out\n",
        z->total_out + (q >= s->read ? q - s->read :
            (s->end - s->read) + (q - s->window))));
    s->mode = s->last ? DRY : TYPE;
    break;
case TABLE:
    NEEDBITS(14)
    s->sub.trees.table = t = (uInt)b & 0x3fff;
#endif
#ifdef PKZIP_BUG_WORKAROUND

```

```

if ((t & 0x1f) > 29 || ((t > 5) & 0x1f) > 29)
{
    s->mode = BAD;
    z->msg = (char*)"too many length or distance symbols";
    r = Z_DATA_ERROR;
    LEAVE
}
#endif
t = 258 + (t & 0x1f) + ((t >> 5) & 0x1f);
if ((s->sub.trees.blens = (uIntf*)ZALLOC(z, t, sizeof(uInt))) == Z_NULL)
{
    r = Z_MEM_ERROR;
    LEAVE
}
DUMPBITS(14)
s->sub.trees.index = 0;
Tracev((stderr, "inflate:          table sizes ok\n"));
s->mode = BTREE;
case BTREE:
    while (s->sub.trees.index < 4 + (s->sub.trees.table >> 10))
    {
        NEEDBITS(3)
        s->sub.trees.blens[border[s->sub.trees.index++]] = (uInt)b & 7;
        DUMPBITS(3)
    }
    while (s->sub.trees.index < 19)
        s->sub.trees.blens[border[s->sub.trees.index++]] = 0;
    s->sub.trees.bb = 7;
    t = inflate_trees_bits(s->sub.trees.blens, &s->sub.trees.bb,
                          &s->sub.trees.tb, s->hufts, z);
    if (t != Z_OK)
    {
        ZFREE(z, s->sub.trees.blens);
        r = t;
        if (r == Z_DATA_ERROR)
            s->mode = BAD;
        LEAVE
    }
    s->sub.trees.index = 0;
    Tracev((stderr, "inflate:          bits tree ok\n"));
    s->mode = DTREE;
case DTREE:
    while (t = s->sub.trees.table,
           s->sub.trees.index < 258 + (t & 0x1f) + ((t >> 5) & 0x1f))
    {
        inflate_huft *h;
        uInt i, j, c;

        t = s->sub.trees.bb;
        NEEDBITS(t)
        h = s->sub.trees.tb + ((uInt)b & inflate_mask[t]);
        t = h->bits;
        c = h->base;
        if (c < 16)
        {
            DUMPBITS(t)
            s->sub.trees.blens[s->sub.trees.index++] = c;
        }
        else /* c == 16..18 */
        {
            i = c == 18 ? 7 : c - 14;
            j = c == 18 ? 11 : 3;
            NEEDBITS(t + i)
            DUMPBITS(t)
            j += (uInt)b & inflate_mask[i];
            DUMPBITS(i)
            i = s->sub.trees.index;
            t = s->sub.trees.table;
            if (i + j > 258 + (t & 0x1f) + ((t >> 5) & 0x1f) ||
                (c == 16 && i < 1))
            {
                ZFREE(z, s->sub.trees.blens);
                s->mode = BAD;
            }
        }
    }

```



```

        z->msg = (char*)"id bit length repeat";
        r = Z_DATA_ERROR;
        LEAVE
    }
    c = c == 16 ? s->sub.trees.blens[i - 1] : 0;
    do {
        s->sub.trees.blens[i++] = c;
    } while (--j);
    s->sub.trees.index = i;
}
s->sub.trees.tb = Z_NULL;
{
    uInt bl, bd;
    inflate_huft *tl, *td;
    inflate_codes_statef *c;

    bl = 9;          /* must be <= 9 for lookahead assumptions */
    bd = 6;          /* must be <= 9 for lookahead assumptions */
    t = s->sub.trees.table;
    t = inflate_trees_dynamic(257 + (t & 0x1f), 1 + ((t >> 5) & 0x1f),
                             s->sub.trees.blens, &bl, &bd, &tl, &td,
                             s->hufts, z);
    ZFREE(z, s->sub.trees.blens);
    if (t != Z_OK)
    {
        if (t == (uInt)Z_DATA_ERROR)
            s->mode = BAD;
        r = t;
        LEAVE
    }
    Tracev((stderr, "inflate:      trees ok\n"));
    if ((c = inflate_codes_new(bl, bd, tl, td, z)) == Z_NULL)
    {
        r = Z_MEM_ERROR;
        LEAVE
    }
    s->sub.decode.codes = c;
}
s->mode = CODES;
case CODES:
    UPDATE
    if ((r = inflate_codes(s, z, r)) != Z_STREAM_END)
        return inflate_flush(s, z, r);
    r = Z_OK;
    inflate_codes_free(s->sub.decode.codes, z);
    LOAD
    Tracev((stderr, "inflate:      codes end, %lu total out\n",
                    z->total_out + (q >= s->read ? q - s->read :
                    (s->end - s->read) + (q - s->window))));
    if (!s->last)
    {
        s->mode = TYPE;
        break;
    }
    s->mode = DRY;
case DRY:
    FLUSH
    if (s->read != s->write)
        LEAVE
    s->mode = DONE;
case DONE:
    r = Z_STREAM_END;
    LEAVE
case BAD:
    r = Z_DATA_ERROR;
    LEAVE
default:
    r = Z_STREAM_ERROR;
    LEAVE
}
}

```

```

int inflate_blocks_free(s, z)
inflate_blocks_statef *s;
z_stream z;
{
    inflate_blocks_reset(s, z, Z_NULL);
    ZFREE(z, s->window);
    ZFREE(z, s->hufts);
    ZFREE(z, s);
    Tracev((stderr, "inflate:   blocks freed\n"));
    return Z_OK;
}

```

```

void inflate_set_dictionary(s, d, n)
inflate_blocks_statef *s;
const Bytef *d;
uInt n;
{
    zmemcpy(s->window, d, n);
    s->read = s->write = s->window + n;
}

```

```

/* Returns true if inflate is currently at the end of a block generated
 * by Z_SYNC_FLUSH or Z_FULL_FLUSH.
 * IN assertion: s != Z_NULL
 */

```

```

int inflate_blocks_sync_point(s)
inflate_blocks_statef *s;
{
    return s->mode == LENS;
}

```

```

/* infblock.h -- header to use block.c
 * Copyright (C) 1995-1998 Mark Adler
 * For conditions of distribution and use, see copyright notice in zlib.h
 */

```

```

/* WARNING: this file should *not* be used by applications. It is
part of the implementation of the compression library and is
subject to change. Applications should only use zlib.h.
*/

```

```

struct inflate_blocks_state;
typedef struct inflate_blocks_state FAR inflate_blocks_statef;

```

```

extern inflate_blocks_statef * inflate_blocks_new OF((
    z_streamp z,
    check_func c,          /* check function */
    uInt w));              /* window size */

```

```

extern int inflate_blocks OF((
    inflate_blocks_statef *,
    z_streamp ,
    int));                /* initial return code */

```

```

extern void inflate_blocks_reset OF((
    inflate_blocks_statef *,
    z_streamp ,
    uLongf *));           /* check value on output */

```

```

extern int inflate_blocks_free OF((
    inflate_blocks_statef *,
    z_streamp));

```

```

extern void inflate_set_dictionary OF((
    inflate_blocks_statef *s,
    const Bytef *d, /* dictionary */
    uInt n));        /* dictionary length */

```

```

extern int inflate_blocks_sync_point OF((
    inflate_blocks_statef *s));

```

```

        break;
    }
    if (e & 32)                /* end of block */
    {
        Tracevv((stderr, "inflate:         end of block\n"));
        c->mode = WASH;
        break;
    }
    c->mode = BADCODE;         /* invalid code */
    z->msg = (char*)"invalid literal/length code";
    r = Z_DATA_ERROR;
    LEAVE
case LENEXT:                  /* i: getting length extra (have base) */
    j = c->sub.copy.get;
    NEEDBITS(j)
    c->len += (uInt)b & inflate_mask[j];
    DUMPBITS(j)
    c->sub.code.need = c->dbits;
    c->sub.code.tree = c->dtree;
    Tracevv((stderr, "inflate:         length %u\n", c->len));
    c->mode = DIST;
case DIST:                   /* i: get distance next */
    j = c->sub.code.need;
    NEEDBITS(j)
    t = c->sub.code.tree + ((uInt)b & inflate_mask[j]);
    DUMPBITS(t->bits)
    e = (uInt)(t->exop);
    if (e & 16)               /* distance */
    {
        c->sub.copy.get = e & 15;
        c->sub.copy.dist = t->base;
        c->mode = DISTEXT;
        break;
    }
    if ((e & 64) == 0)        /* next table */
    {
        c->sub.code.need = e;
        c->sub.code.tree = t + t->base;
        break;
    }
    c->mode = BADCODE;         /* invalid code */
    z->msg = (char*)"invalid distance code";
    r = Z_DATA_ERROR;
    LEAVE
case DISTEXT:                /* i: getting distance extra */
    j = c->sub.copy.get;
    NEEDBITS(j)
    c->sub.copy.dist += (uInt)b & inflate_mask[j];
    DUMPBITS(j)
    Tracevv((stderr, "inflate:         distance %u\n", c->sub.copy.dist));
    c->mode = COPY;
case COPY:                   /* o: copying bytes in window, waiting for space */
#ifdef __TURBOC__ /* Turbo C bug for following expression */
    f = (uInt)(q - s->window) < c->sub.copy.dist ?
        s->end - (c->sub.copy.dist - (q - s->window)) :
        q - c->sub.copy.dist;
#else
    f = q - c->sub.copy.dist;
    if ((uInt)(q - s->window) < c->sub.copy.dist)
        f = s->end - (c->sub.copy.dist - (uInt)(q - s->window));
#endif
    while (c->len)
    {
        NEEDOUT
        OUTBYTE(*f++)
        if (f == s->end)
            f = s->window;
        c->len--;
    }
    c->mode = START;
    break;
case LIT:                   /* o: got literal, waiting for output space */
    NEEDOUT

```

```

    OUTBYTE(c->sub.lit)
    c->mode = START;
    break;
case WASH:          /* o: got eob, possibly more output */
    if (k > 7)       /* return unused byte, if any */
    {
        Assert(k < 16, "inflate_codes grabbed too many bytes")
        k -= 8;
        n++;
        p--;        /* can always return one */
    }
    FLUSH
    if (s->read != s->write)
        LEAVE
    c->mode = END;
case END:
    r = Z_STREAM_END;
    LEAVE
case BADCODE:       /* x: got error */
    r = Z_DATA_ERROR;
    LEAVE
default:
    r = Z_STREAM_ERROR;
    LEAVE
}
#ifdef NEED_DUMMY_RETURN
return Z_STREAM_ERROR; /* Some dumb compilers complain without this */
#endif

void inflate_codes_free(c, z)
inflate_codes_statef *c;
z_streamp z;
{
    ZFREE(z, c);
    Tracev((stderr, "inflate:      codes free\n"));
}

```